

BOLT BERANEK AND NEWMAN INC

CONSULTING · DEVELOPMENT · RESEARCH

AFCRL-68-0381

AD675553

THE BBN HYBRID PROCESSOR

Theodore R. Strollo  
Raymond S. Tomlinson  
Edward R. Fiala  
Jerome I. Elkind

Bolt Beranek and Newman Inc  
50 Moulton Street  
Cambridge, Massachusetts 02138

Contract No. F19628-68-C-0125  
Project No. 8668  
Task No. 866800  
Work Unit No. 86680001

Scientific Report No. 7

This research was sponsored by the Advanced Research Projects Agency under ARPA Order No. 527

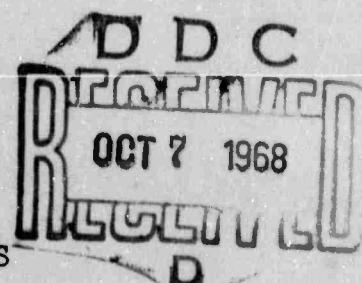
30 June 1968

Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce, for sale to the general public.

Contract Monitor: Hans Zschirnt  
Data Sciences Laboratory

Prepared for:

AIR FORCE CAMBRIDGE RESEARCH LABORATORIES  
OFFICE OF AEROSPACE RESEARCH  
UNITED STATES AIR FORCE  
BEDFORD, MASSACHUSETTS 01730



Reproduced by the  
CLEARINGHOUSE  
for Federal Scientific & Technical  
Information Springfield Va. 22151

## THE BBN HYBRID PROCESSOR

Theodore R. Strollo  
Raymond S. Tomlinson  
Edward R. Fiala  
Jerome I. Elkind

Bolt Beranek and Newman Inc  
50 Moulton Street  
Cambridge, Massachusetts 02138

Contract No. F19628-68-C-0125  
Project No. 8668  
Task No. 866800  
Work Unit No. 86680001

Scientific Report No. 7

This research was sponsored by the Advanced Research Projects Agency under ARPA Order No. 627

30 June 1968

Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce, for sale to the general public.

Contract Monitor: Hans Zschirnt  
Data Sciences Laboratory

Prepared for:

AIR FORCE CAMBRIDGE RESEARCH LABORATORIES  
OFFICE OF AEROSPACE RESEARCH  
UNITED STATES AIR FORCE  
BEDFORD, MASSACHUSETTS 01730

Abstract

Real-time experiments present special problems to a time-shared computer system. For example, simulations, waveform analysis, waveform generations, and displays require very high-rate, closely timed input/output between the digital computer's core memory and external devices. In many cases, conventional input/output methods fail to satisfy the requirements of the experiment. For this reason, Bolt Beranek and Newman Inc., has developed the concept of a special purpose "Hybrid Processor" to perform real-time input/output.

This report describes the hardware and software for the Hybrid Processor which has been constructed for BBN's SDS-940 computer, and the revisions planned for the Hybrid Processor which will be designed for the PDP-10.

## TABLE OF CONTENTS

	<u>Page</u>
Abstract	1
List of Figures and Tables	iii
1. INTRODUCTION	1
2. REAL-TIME EXPERIMENTS	2
3. CONVENTIONAL REAL-TIME INPUT/OUTPUT	4
4. THE HYBRID PROCESSOR CONCEPT	7
4.1 Separation of I/O from Computation	7
4.2 A Hybrid Processor	7
4.3 Implementation of the Hybrid Processor on the SDS-940	7
4.4 Accurately Timing Hybrid Input/Output	12
4.5 Scheduling Hybrid I/O with a Hybrid Processor	15
4.6 Control of Hybrid Processor	16
5. REAL-TIME CPU USAGE	17
6. FUTURE HYBRID PROCESSOR REVISIONS	20
6.1 New Clock System	20
6.2 Hybrid I/O via the Memory Channel Facility	21
6.3 Protection of Channel Hybrid I/O	28
6.4 Direct Input/Output Facility	29
6.5 Typical Division between Channel and Direct I/O	30
7. CONCLUSIONS	32
APPENDICES	
APPENDIX A      Hardware Specs	33
APPENDIX B      Hybrid Processor Software	53
APPENDIX C      Hardware Patch Panel	81
REFERENCES	88

## LIST OF FIGURES AND TABLES

	<u>Page</u>
Fig. 4.1 Command and Data Tables	8
Fig. 4.2 Hybrid Processor Command Format	11
Fig. 6.1 Command and Data Table Structures	21
Fig. 6.2 Command Format	22
Fig. 6.3 Flow Tables	25
Fig. 6.4 Command and Data Flow Words	26
Table A-1 System Mode EOM's	33
Fig. A-2 Location of Interlace Words	41
Table A-3 Device Types	42
Table A-4 BBN 940 Interrupts	45
Fig. C-1 Photograph of Patch Panel	81
Fig. C-2 Photograph of Patching to a Sample and Hold Input	84
Fig. C-3 Photograph of Conventional Display Patching	85

## 1. INTRODUCTION

While a great deal of work has been done to time-share a computer system with several users at teletype terminals, time-sharing a computer for real-time experiments is a relatively recent undertaking. The demands a real-time experiment presents to the digital computer are quite a bit different from the demands of a teletype terminal. An experiment often requires precisely timed data transfers. Some experiments require lengthy computations between these data transfers. However, a single real-time experiment may use only a small portion of the available real world devices and a fraction of the total computation services; so the time sharing of the system's resources to handle multiple real-time experiments is desirable.

Previous implementations of real-time experiment capabilities on a digital computer system have generally handled only the class of synchronous real-time demands with all real-time I/O performed directly by the central processing unit. Examples of such implementations are the DEC PDP-6 hybrid system at United Aircraft (Ref. 1) and the DEC PDP-1 hybrid system at the M.I.T. Electronic Systems Laboratory (Ref. 2,3).

We shall explain in this report the distinct advantages of separating the real-time computations from the real-time I/O and performing the real-time computations on the central processor while performing the real-time I/O on a special processor.

We call our implementation of this special processor for real-time I/O a Hybrid Processor. This division of labor between two processors permits one to handle both synchronous and asynchronous real-time demands and to guarantee service to more real-time demands than is possible without such a division.

## 2. REAL-TIME EXPERIMENTS

When we speak of real-time demands, one generally visualizes applications where the computer system is required to interact with an experiment which is being conducted in the real world in real-time. While these experiments are certainly examples of real-time demands, we extend the definition to include any external demands of the computer system where the reaction time to a demand must be relatively small. As this rapid response becomes more critical, the demand is considered more "real-time."

The first demand which fits the real-time category is the data acquisition or data generation problem. Here the computer system is called upon to sample repetitively an analog signal or to generate an analog waveform. The data rates can get as high as ~20 KHz for speech sampling or generation. Sampling or generation is generally performed periodically at a fixed rate; thus, most of these problems may be considered synchronous. However, in some cases the initiation of the sampling is not controllable by the computer system. For example, the sampling of a recording on an analog magnetic tape may have to start when a control signal is read off the tape. The computer system generally has no control over this start time, and this sampling problem is now asynchronous to the computer system.

Simulation problems are also in the real-time category. Generally an analog computer will perform the linear portions of the simulation while the digital computer will perform the non-linear portions. The digital portion of the system exhibits synchronous behavior characterized by:

- 1) Sampling a set of input values from the analog computer at a specific time.
- 2) Performing some digital computation utilizing these input values.
- 3) Setting up some output values for the analog computer at a specific time.

Display generation problems are often in the real-time category. This is especially true when displays are used to simulate real world activity such as the view of a runway while landing an aircraft or the view of the instruments of a control panel. Display generation problems are much like any waveform generation problem, but the data rate is generally high, and display problems often tend to present asynchronous demands such as the generation of a new display in response to the changes in a manual control by a human operator.

### 3. CONVENTIONAL REAL-TIME I/O

There are two conventional methods of synchronizing CPU performed I/O to real-time demands:

Method 1: initiate the I/O in the user's program and synchronize the running of the user's program to the real-time I/O demands. This initiation of I/O could be performed by a monitor call or by enabling the user to issue direct I/O commands to his devices. The latter is faster but requires hardware device protection.

Method 2: use a clock pulse to trigger a CPU interrupt routine which will perform the real-time I/O independent of the running of the user's program.

Both of these methods have similar problems because utilizing a CPU to perform real-time I/O consumes a great deal of its capacity on a task for which it was not designed. A general purpose processor is not well-suited to performing real-time I/O because it cannot switch its attention between tasks rapidly enough. Switching times on the order of a half-millisecond for user processes and 20 microseconds for interrupt routine switches are typical of conventional time-shared general purpose processors. This means the overhead of performing real-time I/O utilizing the CPU is quite prohibitive, especially if the real-time I/O rate is high.

A general purpose processor is not well-suited to accurate timing control. Since current central processors are generally asynchronous machines, it is difficult to time operations precisely with a central processor. Further timing skew is introduced by the tendency of general purpose time-sharing systems to get

themselves into non-interruptible states for short periods such as by disabling the interrupt system or by processing higher priority interrupts or by interrupting out of the user process's run period.

It would seem that processing all real-time I/O on the highest priority interrupt channel would be the most desirable method for CPU performed real-time I/O since this seems to minimize skew and overhead. However, one finds it is impossible to make every device the "highest priority" on the interrupt system, and, in fact, there may be other system constraints which force the real-time I/O to be on a lower priority interrupt channel.

Another limitation of interrupt driven real-time I/O is that, in general, one interrupt request will initiate several real-time I/O data transfers. This means the time spent in the interrupt service routine for performing the real-time I/O varies with the number of data transfers to be performed. These interrupt requests will tend to queue up unless their interrupt service routines are scheduled to be non-overlapping.

The precise scheduling of both interrupt-driven and user-initiated real-time I/O is very difficult unless all of the real-time I/O demands are exactly periodic and synchronized to the clock system of the CPU. The source of this difficulty is the inherently slow reaction time of a general purpose processor to random requests for service. If all requests are guaranteed to be exactly periodic they can be scheduled in advance, and the CPU can be prepared to process them.

We conclude that when real-time I/O is to be performed with precise timing, when the degradation of CPU performance from CPU driven real-time I/O cannot be tolerated, or when both synchronous and asynchronous demands must be handled, real-time I/O should not be performed by the conventional CPU driven methods.

#### 4. THE HYBRID PROCESSOR CONCEPT

##### 4.1. Separation of I/O from Computation

Considering the examples given in Section 2 carefully, it is evident that most real-time problems place very stringent demands on I/O timing, but less stringent demands on the related CPU computation. That is, the sampling of input data and the conversion of output data to analog signals must be done at precise times, whereas the computation may be done any time after the input operation provided that the output data is ready before it is converted to analog signals. In addition, the I/O operations do not require the full talent of the CPU, but do require better timing accuracy than what the CPU is capable of providing (due to interrupts, etc.). Therefore, it seems feasible and in fact desirable, to separate the I/O from the computation and provide separate hardware.

##### 4.2. A Hybrid Processor

Our system performs hybrid I/O through a device which we call a Hybrid Processor. This device is a special purpose processor with the following important characteristics:

- 1) Hybrid I/O is performed directly between core memory and hybrid devices.
- 2) The Hybrid Processor is multiplexed among several "processes" thus allowing several independent concurrent hybrid interactions.
- 3) The switching time between processes is very small (~100 nsec) because the information about the state of a process is small and can be changed rapidly.

- 4) The time required to perform a single hybrid I/O transfer is kept very short (~20  $\mu$ sec) because of the special purpose nature of the processor.
- 5) All hybrid I/O interactions are handled in a uniform manner thus requiring no hardware or software changes to incorporate a new hybrid I/O device.

#### 4.3 Implementation of the Hybrid Processor on the SDS-940

The Hybrid Processor functionally resembles a multiplexor I/O processor. The processor multiplexes its service to 4 hybrid I/O processes. (Hardware can be expanded to 6 processes.) The commands and data for each process are kept in blocks (tables) in core memory. Each hybrid I/O process is identified by its own command and data tables. While servicing a given process, the Hybrid Processor usually operates sequentially on the process's tables; a command is picked up from the command table, then a data word is operated on from the data table, then the current pointers into both tables are incremented by one for the next service.

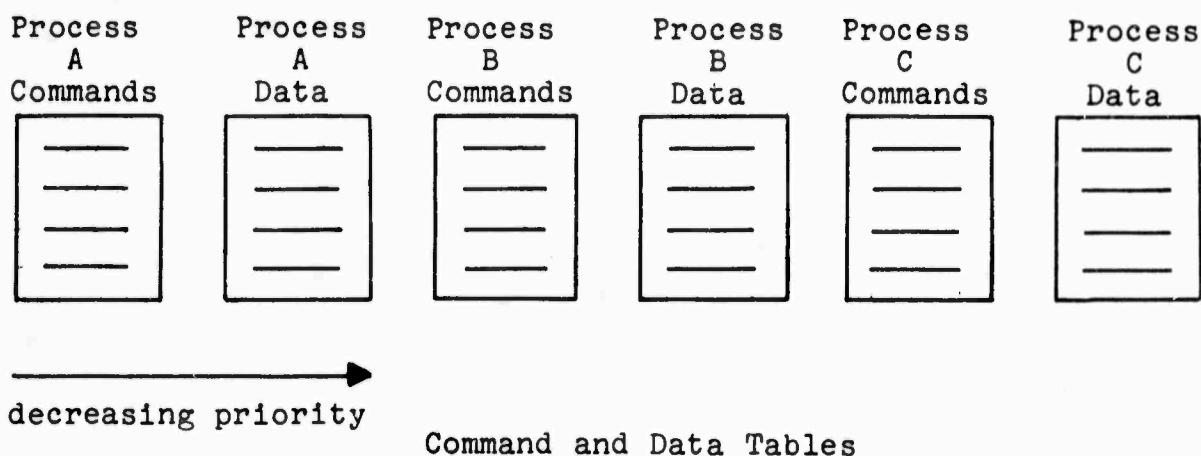


Figure 4.1

Figure 4.1 shows the identification of each process with a separate command and data table. These processes, in general, run completely independently with arbitrary rates. Therefore, it is quite likely that more than one process may be using the Hybrid Processor at any one time. The ability of our Hybrid Processor to switch its attention between various processes on a priority basis is one of its most important features. This switching is accomplished by associating with each process a pre-established priority. When the Hybrid Processor completes the execution of a command for a given process, the queue of all demands for service is considered. The highest priority demand in this queue is then serviced.

#### 4.3.1 Connection of Hybrid Processor to SDS-940

The Hybrid Processor is attached to the SDS-940 via a Data Multiplexor Channel (DMC) with a modified Data Sub-Channel II (DSC II). The status of a block transfer for a DMC subchannel is normally contained in one of two "internal interlace words" which are located in fixed adjacent positions in core memory. These interlace words contain the remaining word count in the leftmost bits and current location of the block transfer in the rightmost bits. The economy of using core memory words instead of flip-flop registers is quite important and this economy is retained by the Hybrid Processor. However, the DSC II has been modified so that the locations of these interlace words are no longer fixed but are uniquely determined by the particular process selected for service. That is, Process A uses words  $n,* n+1$  as interlace words; Process B uses words

---

\* where n is a memory address which is  $\neq$  MCD 4

$n+4$ ,  $n+5$ , ... Also, during the command fetch, the even interlace word is used, while during the data fetch or store, the odd interlace word is used. Note that as before the state of the block transfers is completely contained in the memory interlace words.

Each hybrid process can switch command and data tables by using the "cycle" interlace words. Every process actually has four interlace words. Words  $n$  and  $n+1$  are the current command and data table interlace words respectively. Words  $n+2$  and  $n+3$  are the cycle command and data table interlace words. A process can cause the contents of its cycle interlace words to be moved into the current interlace words. This effectively switches the command and/or data tables to new core areas.

#### 4.3.2 Requesting Service from the Hybrid Processor

It is necessary for each hybrid I/O process to have a means for indicating to the Hybrid Processor that it needs service. In addition, the process must state when it no longer requires service. The latter is especially important for the high priority processes which could lock out the lower priority processes by continuously demanding service. The scheme for requesting service is quite simple. For each process there is a request flip-flop which is set when service is required. When the Hybrid Processor executes a command for this process, the process request flip-flop is cleared. No additional service will be given to this process until its request flip-flop is set again. It is the duty of each process to set its request flip-flop every time it wants a command executed.

The process request flip-flop is set in one of three ways:

1. By a clock which is enabled for the process.
2. By an external signal which is associated with the process.
3. By the current command for this process specifying the next command is to be executed as soon as possible after the current command is completed.

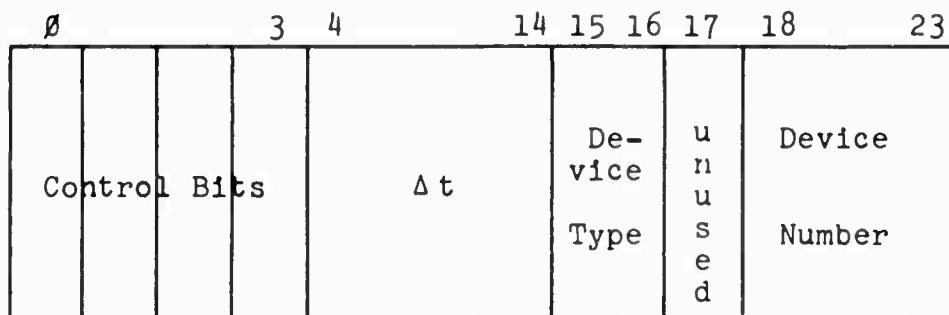
Processes remove themselves from the Hybrid Processor's service queue by one of the following:

1. Specifying a non-zero time between command executions.
2. Reaching the end of a command sequence.
3. Specifying a wait for external signal before proceeding.

These conditions inhibit the setting of the process request flip-flop.

#### 4.3.3 Hybrid Processor Commands

Every Hybrid Processor command in a command table has the format of Figure 4.2.



HYBRID PROCESSOR COMMAND FORMAT

Figure 4.2

The device type field specifies one of the 4 possible hybrid device types, including A/D converters, D/A converters, D/D inputs, D/D outputs. The device number field specifies a particular device within a device type.

The  $\Delta t$  field specifies the time between the execution of this command and the next command in the table. The uses of the control bits are described in Appendix A.

#### 4.4 Accurately Timing Hybrid Input/Output

##### 4.4.1 System Clocks and Tickers

Our computer system has three kinds of clocks: a time of day clock, an alarm clock, and process clocks.

###### Time of Day Clock

All of the clocks tick at the same 100 kHz crystal controlled rate. The use of a common tick pulse eliminates the necessity of any conversion of counts between any of the clocks and keeps them all synchronized.

The time of day clock is an 11-bit clock which counts from 0 to 1999, and then repeats. The transition from 1999 to 0 generates an interrupt request to the CPU. Since the time of day clock can be read into the 940 at any time, this clock can be used to measure elapsed time at a resolution of 10  $\mu$ sec. The interrupts generated by the time of day clock are counted by the software as high-order clock bits. These interrupts occur every 20 milliseconds.

Alarm Clock

The alarm clock holds a 12-bit count which is loaded directly from the CPU. This clock is a down counter which generates an interrupt request when the clock count is zero. This same count equal to  $\emptyset$  condition initiates each hybrid process which is enabled for starting by the alarm clock. The reasons this condition generates a CPU interrupt request are:

- 1) The CPU can establish a new alarm clock count in the interrupt routine.
- 2) The alarm clock can be used to notify the CPU that a computation process requires attention at this time.

Since the alarm clock interrupt is of lower priority than some of the SDS-940 internal interrupts, we cannot guarantee immediate response to the alarm clock interrupt request. Therefore, the alarm clock count and its count  $\emptyset$  enable bits are double-buffered which means the alarm clock  $\emptyset$  condition causes a new count and set of process enable conditions to be strobed into the active alarm clock.

Process Clocks

The commands to the hybrid devices will not be generated simultaneously but sequentially as the commands are fetched from core. In fact, since several processors (CPU, Hybrid Processor, Drum Processor,...) are competing for the same core memory, it is not possible to determine the exact times between command fetches. Therefore, a positive synchronization scheme for hybrid I/O has been developed. This scheme utilizes the process clocks associated with the Hybrid Processor.

Two of our hybrid I/O processes have independent 12-bit clocks. These two processes are called clocked processes and make use of the  $\Delta t$  field of a command word in the following ways:

- 1) The clock is a down counter counting at 100 kHz. When the clock count goes from 0 to -1, a request for hybrid I/O service is made.
- 2) When a command for a clocked process is executed, the value  $\Delta t$  is added to the contents of the process clock. This addition is required since several "ticks" may elapse before the command is actually executed, and it is necessary to account for these elapsed ticks if the average time between command executions is to be correct.
- 3) When the clock count goes from 0 to -1, a synchronization pulse is generated which may be externally patched to devices which require precise timing sync. These include sample and hold gates, rank 1 to rank 2 updates on dual rank D/A converters, and other devices requiring sync pulses. Note that these sync pulses occur precisely when the  $\Delta t$  between commands elapses.

The remaining two hybrid I/O processes do not have clocks and are called unclocked processes. These processes are used for low-speed hybrid I/O demands. These low-speed processes are initiated by the system alarm clock. Once a low-speed process is started, data interactions will occur as rapidly as possible (until a halt condition is reached), but may be pre-empted by requests from the higher priority processes with independent process clocks.

Unclocked processes do not use the  $\Delta t$  field for timing. The unclocked processes were implemented primarily because they are much less expensive than the clocked processes.

#### 4.5 Scheduling Hybrid I/O with a Hybrid Processor

Suppose the maximum time to service a hybrid I/O request were  $\alpha$ . This time would be measured from when the Hybrid Processor started processing a request until it finished this request and could begin working on another. Then a conservative estimate of the bandwidth of the Hybrid Processor would be  $1/\alpha$ .

A simple scheduling technique would involve selling fractions of this bandwidth to users. Each user would buy  $1/\alpha_i$  of the bandwidth of the Hybrid Processor such that

$$\sum_{\text{all } i} \frac{1}{\alpha_i} \leq \frac{1}{\alpha}$$

It would now be necessary for the Hybrid Processor (or whatever controls the Hybrid Processor) to insist that user  $i$  never perform hybrid I/O faster than  $\alpha_i$  time between hybrid interactions. This simple check could be performed by either hardware or software. This technique would guarantee that the Hybrid Processor never be over-committed.

On the SDS-940 implementation of the Hybrid Processor, the maximum  $\alpha$  is approximately 20  $\mu$ sec.\* This means the Processor has a maximum bandwidth of 50 kHz.

---

\* For D/A conversion, ~30  $\mu$ sec for A/D conversions

#### 4.6 Control of Hybrid Processor

The actual control of each Hybrid Process is done by the time-sharing monitor.

##### 4.6.1 Device and Timing Protection

Note that the Hybrid Processor commands reference all Hybrid Devices in an unrestricted manner. If the user is given direct access to these commands, he could detrimentally affect another user's experiment by changing a value on another user's D/A converter, for example. Also, one user could easily lock out Hybrid Processor service from another user if he had a higher priority process and usurped all of the Hybrid Processor's capacity. It is therefore not possible for the user to construct his own command tables. Instead, the time-sharing monitor constructs these tables for the user and keeps them in its own core. The monitor makes certain that a user does not access another user's devices or usurp all of the Hybrid Processor time.

## 5. REAL-TIME CPU USAGE

With the availability of a Hybrid Processor I/O feature, it is no longer necessary that either user programs or user I/O be periodic. The I/O can be precisely timed using the Hybrid Processor independent of CPU activity. Therefore, it no longer is necessary to start CPU computation at exact times.

With the Hybrid Processor the following characteristics of real-time computation processes become reasonable:

Suppose that for each process the following parameters were specified:

- 1) T The period of the process (exact period if synchronous, minimum period if asynchronous).
- 2) P The maximum amount of CPU time the process may require each period.
- 3) D The maximum tolerable delay between the moment the process requests service and the time when all servicing has been completed (most synchronous processes would allow service to be completed any time during the period, i.e.,  $D = T$ ).
- 4) Whether the process is synchronous or asynchronous.

Using this characterization, the demand of the process upon the system might be phrased as follows: "When my process requests service it must be granted P seconds of CPU time within D seconds of when the request is made." The parameters T and knowledge of whether or not the process is synchronous enables the system to decide whether the demands of this process (and all others) can be successfully met.

The system cannot, of course, guarantee service to a set of real-time processes with arbitrary P's, D's, and T's. In fact, two restrictions are obvious:

$$1) \emptyset < P_i < D_i \leq T_i \text{ and}$$

$$2) \sum_{\text{all } i \text{ processes}} \frac{P_i}{T_i} \leq 1$$

If the sum in (2) were greater than unity, it would be possible for the real-time processes to require more than 100% of the available CPU time.

The scheduling algorithm used to select which process runs at any time is intimately related to the guarantees which the system can make to a set of users. It would be desirable to find a scheduling algorithm which would allow

$$\sum_{\text{all } i \text{ processes}} \frac{P_i}{T_i} \text{ to be close to 1}$$

and would minimize the amount of switching between processes to reduce overhead. It can be shown that if switching time is negligible, no algorithm can do a better job of scheduling than the following:

Run the process which must be completed soonest.

That is, whenever a process requests service, the system computes the time when the process must complete service ( $\tau_i$ ) which is

equal to the current time plus  $D_i$ . The system then decides to run the process with the minimum  $\tau_i$ . Whenever servicing is completed or aborted (for trying to use more than  $P_i$  CPU time) the system runs next the process with minimum  $\tau_i$ . This algorithm and the necessary and sufficient conditions under which the system can undertake to run a set of processes are discussed in detail in another document (Ref. 4).

## 6. FUTURE HYBRID PROCESSOR REVISIONS

Several changes will be made in our new hybrid I/O system for our next research computer (a DEC PDP-10). These changes will increase the total available bandwidth, improve the command/data flow control so that even less CPU capacity will be required to direct the Hybrid Processor, make several improvements to the clock system, etc.

### 6.1 New Clock System

A 36-bit time of day clock will be implemented which counts at 100 kHz. It will be possible to read this time via DATAI over the PDP-10 I/O Buss. This 36-count will recycle approximately every 8 days.

The carry input to each successive stage of the time of day clock will be counting at half the frequency of the previous stage. This gives us a convenient source of clock pulses at 100 kHz, 50 kHz, 25 kHz,... These would be amplified and made available to users for synchronization purposes.

At least two 36-bit "alarm" registers will be used in conjunction with the clock. These registers will be compared with the values in the clock after each "tick" settles down. If a match on any register is found, the following events will occur:

- A) A CPU interrupt request will be generated so that a new 36-bit value may be placed in the alarm register and any CPU action which was to be initiated at this time will be triggered (such as the scheduling of a new process to run).

B) Each alarm register will have 8 enable bits whose set output will be gated with the alarm pulse and this gated result will be buffered and available for patching to trigger external devices.

The control of the alarm registers will require the use of a PDP-10 CONO instruction to the selected alarm register to set any combination of the 8 enable bits followed by a PDP-10 DATA0 instruction to the selected alarm register to set up the 36 bits of the register itself.

## 6.2 Hybrid I/O via the Memory Channel Facility

### 6.2.1 Hybrid I/O Channel Command Format

The channel I/O capability will be quite similar to the capability of our current SDS 940 Hybrid Processor. The channel will operate on command and data tables with each command paired with a corresponding word in the data table. (see Figure 6.1)

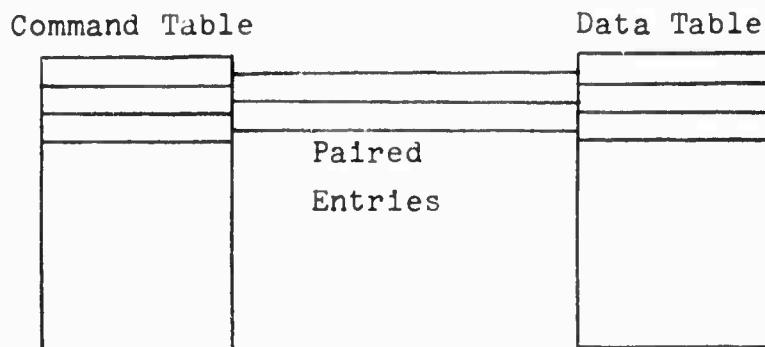


Figure 6.1 Command and Data Table Structure

The command format for the 36-bit PDP-10 memory word is shown in Figure 6.2.

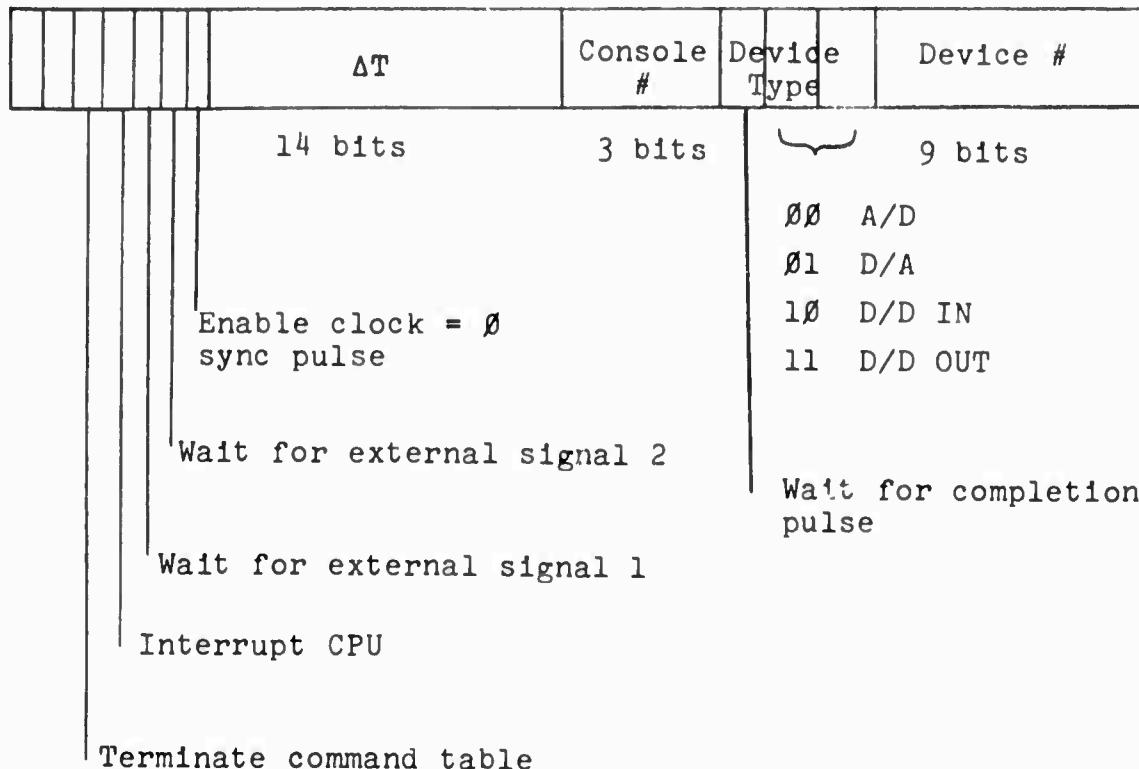


Figure 6.2 Command Format

The various fields of each command are explained below:

- A) The Console Number field selects one of the 8 hybrid consoles each of which will have its own set of hybrid devices including A/D converters, D/A converters and discretes.
- B) The Device Type field selects the type of device (A/D, D/A, D/D input, D/D output) and has a bit which specifies whether or not the process must wait for a completion pulse from the device.

- C) The Device Number field selects one of 512 possible devices within a type such as a multiplexor channel on an A/D converter, etc.
- D) The  $\Delta t$  field specifies the time (in 10 usecond increments) between the execution of the current command and the execution of the next command.
- E) The enable clock = 0 sync bit permits the next clock = 0 condition to generate a pulse which is buffered and available for patching to trigger inputs of various devices such as sample and hold gates, update rank 2 inputs of dual rank D/A converters, etc.
- F) The external signal wait bits cause the process to wait for one of two specified external signals after the duration specified by the  $\Delta t$  field has elapsed.
- G) The interrupt CPU bit generates a CPU interrupt request when it is necessary to invoke CPU action. For example, a command may specify a wait for external signal, then the next command may generate an interrupt to activate the user's program in response to the external signal.
- H) A Terminate Command Table bit will specify the last command word in the table.

The format of each paired data word will depend upon the particular device being used. With D/A conversion, for example, the left 15 bits might be used for the output bits to the converter.

The channel I/O capability must include a mechanism for keeping track of where the next command and data words are located in memory. In the 940 Hybrid Processor system 1,2/ the DMC channel's internal interlace feature is used.

The use of core memory locations for this information is quite economical but also quite slow on the 940. Three

memory cycles are required for each word accessed in the table, and the DMC makes matters worse since it can only operate at roughly half the SDS-940 memory bandwidth. We would like to retain the economy of core memory "internal interlace" words on the PDP-10 without the speed sacrifice. Since the PDP-10 core memories provide read-pause-write cycles 3/, the read, increment, and update interlace word can be reduced from 2 memory cycles on the SDS-940 to 1 memory cycle on the PDP-10. This would total 2 memory cycles per word accessed in each table on the PDP-10, that is 4 memory cycles per hybrid interaction. In addition, the PDP-10 channel I/O feature will be implemented with logic capable of running at full memory rate.

#### 6.2.2 Flow of Control for Channel Hybrid I/O

When the end of a table is reached, a decision must be made to determine which table to use next. In the SDS-940 Hybrid Processor two interlace words are used for each process. When the active interlace word is counted down to 0, the contents of the alternate interlace word are used. This procedure is called "cycling" and has the advantage of being economical, but the disadvantage of requiring CPU reloading of the alternate interlace word before the current table is used to completion.

The proposed PDP-10 Hybrid Processor will have a more elaborate flow of control mechanism. There will be a command flow table and a data flow table. (see Figure 6.3)

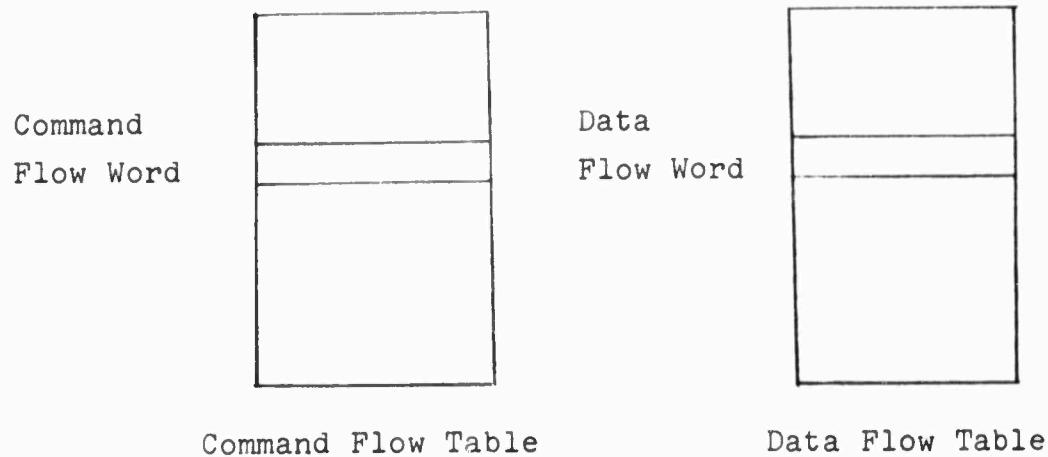


Figure 6.3 Flow Tables

These flow tables will in general be executed in sequential order except where flow words change this order. The command flow and data flow words will have the format displayed in Figure 6.4.

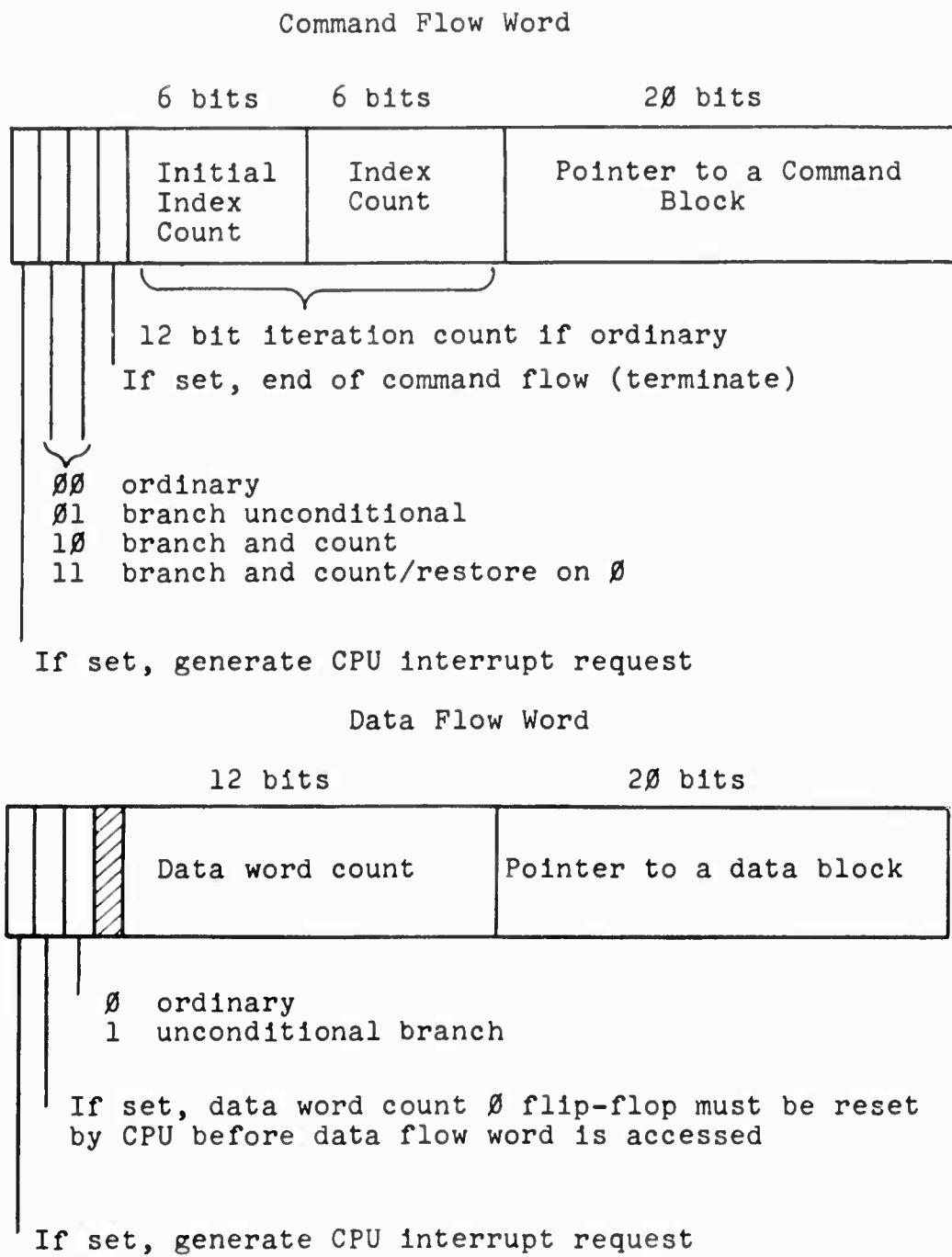


Figure 6.4

Command and Data Flow Words

The ordinary command flow word will specify the beginning of a command table\* and the number of times to iterate that command table. When done with this command table, the channel feature will access the next command flow word in the command flow table. Similarly, the ordinary data flow word will specify the beginning of a data table and the number of words in that table. When done with this data table, the channel feature will access the next data flow word in the data flow table.

The other modes of command flow words will provide for:

- A) Unconditional branch in the command flow table which will cause the next command flow word to be fetched from the specified address.
- B) Branch and count which will decrement the index count field and branch if index count is  $> \emptyset$  or fall through to the next command flow word if index count =  $\emptyset$ .
- C) Branch and Count/Restore will act like Branch and Count except when the index count =  $\emptyset$ , the contents of the initial index count field will be stored in the index count field.

The end of command flow bit is used to terminate a hybrid interaction. The CPU interrupt request bit is used to alert the CPU that a particular place in the command flow has been reached.

The other mode of data flow words will provide for unconditional branching in the data flow table which will cause

---

\* Recall the end of the command table is specified by the terminate bit in the command word.

the next data flow word to be fetched from the specified address. The CPU interrupt request bit is used to alert the CPU that a particular place in the data flow has been reached. The use of the data word count  $\emptyset$  condition test is explained in Appendix B.

#### 6.2.3 Multiple Channel Hybrid I/O Interactions

There will generally be several simultaneous yet independent hybrid interactions which will be handled by the hybrid processor. We again call each of these hybrid interactions a hybrid process. Each process must have a pointer into the command flow table and a pointer into the data flow table. These four pointers per process would be economical if all stored in fixed core memory locations. It would be worthwhile to keep at least the command and data table pointers in flip-flop registers.

#### 6.3 Protection of Channel Hybrid I/O

The simplest way to protect channel I/O is to insist that all command tables be kept in the monitor and checked for validity before execution. If these tables are kept with the user's program, hardware mechanisms must be provided to:

- A) Establish a new set of legal devices to access whenever a new process is selected to run.
- B) Make certain that the minimum  $\Delta t$  specified in a command be greater than the minimum permissible  $\Delta t$  allowed for this process.

These hardware checks are expensive. There are advantages other than cost to keeping command tables in the monitor. The monitor can worry about how often to replicate command tables to keep references to the command flow table minimal. Since command tables are often short, command tables for several processes can generally be kept in one core table instead of keeping a separate table for each process with the associated user's program.

#### 6.4 Direct I/O Facility

##### 6.4.1 Implementation of Direct I/O

A simple and inexpensive method of providing direct hybrid I/O which would be sufficiently protected so that user programs could safely issue direct hybrid I/O commands would involve treating direct I/O just like any other channel hybrid process. The commands and data would be transferred over the I/O buss instead of the memory buss. Two 18 bit registers would be required. One would hold a command issued over the I/O buss until the Hybrid Processor could process this command. The other would hold the output data or would subsequently hold the input data depending on the action the hybrid command invoked. For example, a D/A conversion performed by direct I/O would consist of

DATA0 HP,X

The location X would contain in its right half

3 bits      3 bits      9 bits

console #	device type	device #
-----------	-------------	----------

the device and console selection bits of a normal hybrid

command. The left half of location X would contain the digital value to be sent to the selected D/A converter.

#### 6.4.2 Protection of Direct I/O

Direct I/O will require hardware device protection and timing protection. Device protection is necessary to prevent one user from accessing another user's devices. This protection would be implemented as described by Connally. (Ref.2) Direct I/O timing must be protected to prevent the direct I/O requests from usurping too much of the capacity of the Hybrid Processor which would prevent channel hybrid I/O from running properly. A hardware mechanism must be provided to inhibit direct I/O commands which are spaced too closely. This mechanism would be a counter register containing the minimum  $\Delta t$  permissible between direct I/O commands. Direct I/O commands spaced more closely than the minimum permissible  $\Delta t$  will be trapped.

### 6.5 Typical Division between Channel and Direct I/O

#### 6.5.1 When and How to Use the Channel I/O Facility

The channel I/O feature would be used for all hybrid I/O which have the following characteristics:

- 1) Hybrid I/O which must be performed whether or not the user's program which initiated this I/O is running.
- 2) Hybrid I/O which must be very accurately timed.
- 3) Hybrid I/O which has a repetition period of 20 MS or less (50 Hz or higher). (Since the switching time of the CPU is about 1/2 MS, we do not want users who

require service at this rate or higher to use direct I/O since they would consume more than 2.5% of the CPU time just in switching.)

The user software for accessing the channel I/O would be similar to the software in our SDS 940 time sharing system for the Hybrid Processor. We have included in Appendix B a sample program written for the SDS-940. This is a complete, self-contained operating program. The program repetitively plays back the digitized version of a waveform over a D/A converter channel.

Note that only 25 instructions in this program actually call upon the Hybrid Processor software for performing I/O. These instructions are bracketed to distinguish them from other code. The function of each of these instructions is quite straightforward. These Hybrid Processor instructions assign devices, assign processes, specify table bounds, and start the hybrid process. The other instructions shown in programs are for message output, file manipulation, checking for data overflows and other functions that are only indirectly related to the hybrid process.

#### 6.5.2 When and How to Use the Direct I/O Facility

The direct I/O feature would be used for any hybrid I/O which has the following characteristics:

- 1) Hybrid I/O which will be performed only when the user's program is running, need not be accurately timed, and has a repetition period of more than 20 MS (50 Hz or lower).

2) Hybrid I/O where random selection of hybrid devices is essential but the overhead of setting up command blocks for monitor verification is too time-consuming.

An example of a problem suited to direct I/O would be the setting of servo-controlled pots on an analog computer. The sequence of steps for performing this direct hybrid I/O would involve:

- A) Assigning to the user the device protection level to which he would patch all of his devices.
- B) Telling the monitor the minimum interval between his direct I/O commands. (If the user does not specify this rate, some relatively high value like 1 MS will be assumed.)
- C) The user would now simply issue any PDP-10 commands directed to any of his assigned devices.

## 7. CONCLUSIONS

The use of a Hybrid Processor in our computer system permits us to use our system for many real-time experiments which were not possible in the past, and are not possible on other real-time computer systems. We are able to handle high speed as well as asynchronous hybrid interactions. Most of this is made possible by the separation of the hybrid I/O functions from the computation function. The hybrid I/O functions are performed by a processor specially designed to handle hybrid I/O, and the computations are performed by a general purpose processor.

## APPENDIX A

## A. HARDWARE SPECS

Instruction Interpretation  
(Monitor Mode - Privileged)A.1. Controlling the Hybrid Processor

## A.1.1 EOM's

The system mode EOM's in Table A-1 are used in conjunction with the Hybrid Processor.

TABLE A-1 SYSTEM MODE EOM's

00	Reset the Hybrid Processor
01	Time of Day Clock (read)
02	Load Buffer (Alarm Clock)
03	Load Alarm Clock (via buffer)
04	Load Process Clock A
05	Load Process Clock B
06	Start Process C
07	Clear Process C
10	Start Process D
11	Clear Process D
12	Clear Process A
13	Clear Process B
14	Clear DWCØ A
15	Clear DWCØ B
16	Clear H interrupt A
17	Clear H interrupt B
20	Clear H interrupt C
21	Clear DWCØ C
22	Clear H interrupt D
23	Clear DWCØ D

\*\*\*\*\*

EOM 30000B      I/O Reset

\*\*\*\*\*

I/O reset to the Hybrid Processor. This aborts all current operations and initializes the Hybrid Processor, Time of Day Clock, and Alarm Clock.

\*\*\*\*\*

EOM 30001B      TOD Read

\*\*\*\*\*

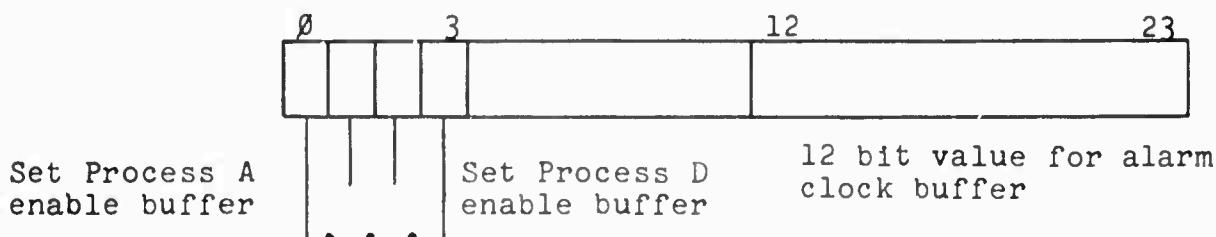
Alert the Time of Day Clock to be read. The next PIN instruction will read in the Time of Day Clock as a right justified 11 bit quantity. The Time of Day Clock counts at 100KH (10  $\mu$ sec between ticks) from 0 to 1999 and then back to 0. A signal is available which is true for an adjustable amount of time and is triggered by transition from the 1999 to the 0 state of the clock. This signal can be used in place of the 60 CPS interrupt and is currently available by a switch option in the 940 interrupt bay.

\*\*\*\*\*

EOM 30002B      Load AC Buffer

\*\*\*\*\*

Load Alarm Clock Buffer. This EOM alerts the Alarm Clock that the next POT will be directed to the Alarm Clock Buffer. The format of the POT word is shown below.



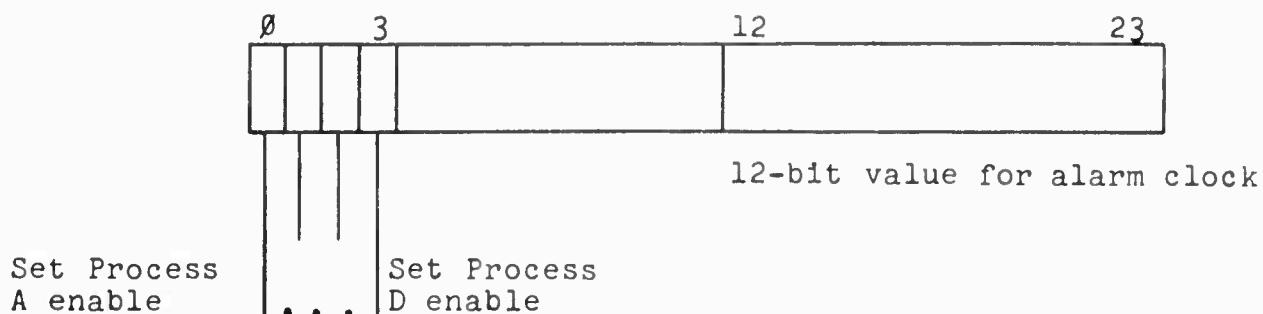
When the Alarm Clock's active register counts down to 0, the 12 bit alarm clock buffer is jammed into the active register and the process enable buffers are jammed into the active process enable flip-flops.

\*\*\*\*\*

EOM 30003B Load AC via Buffer

\*\*\*\*\*

Load the Alarm Clock active register via buffer. The next POT is directed to the Alarm Clock's active register. The POT also destroys the state of the buffer register. The format of the POT word is



The Alarm Clock counts at a 100kHz rate (10  $\mu$ sec between ticks) from the preset value to 0. When the Alarm Clock reaches 0, an interrupt request is generated (location 200B), and a start process pulse is sent to each process (A,B,C, or D) which has its active Alarm Clock enable flip-flop set to 1.

\*\*\*\*\*

EOM 30004B Load Process Clock A

\*\*\*\*\*

The next POT word will preset process Clock A to the value specified in the right most 12 bits and process A will be enabled to run. When the process clock goes from the 0 to the -1 state, a process A request for service is generated and the Hybrid Processor will proceed to serve this process.

\*\*\*\*\*

EOM 30005B Load Process Clock B

\*\*\*\*\*

The next POT word will preset process clock B to the value specified in the right most 12 bits and process B will be enabled to run. When the process clock goes from the 0 to the -1 state, a process B request for service is generated and the Hybrid Processor will proceed to serve this process.

\*\*\*\*\*

EOM 30006B Start Process C

\*\*\*\*\*

Sets unclocked process C's request for service flip-flop and enables process C to run.

\*\*\*\*\*

EOM 30007B Clear Process C

Aborts process C.

\*\*\*\*\*

EOM 30010B Start Process D

\*\*\*\*\*

Sets unclocked process D's request for service flip-flop and enables process D to run.

\*\*\*\*\*

EOM 30011B Clear Process D

Aborts process D.

\*\*\*\*\*

EOM 30012B Clear Process A

Aborts process A.

\*\*\*\*\*  
EOM 30013B Clear Process B

\*\*\*\*\*  
Aborts process B.

\*\*\*\*\*  
EOM 30014B Clear Process A DWCØ

\*\*\*\*\*  
Clears the data word count Ø flip-flop for process A.  
The EOM should be issued every time a data cycle operation occurs  
to permit subsequent data cycling. If a data cycle occurs while  
the process DWCØ flip-flop is set, the process is aborted (note  
the cycle operation does however occur).

\*\*\*\*\*  
EOM 30015B Clear Process B DWCØ

\*\*\*\*\*  
Clears the data word count Ø flip-flop for process B.  
This EOM should be issued everytime a data cycle operation occurs  
to permit subsequent data cycling. If a data cycle occurs while the  
process DWCØ flip-flop is set, the process is aborted (note the  
cycle operation does however occur).

\*\*\*\*\*  
EOM 30016B Clear H interrupt A

\*\*\*\*\*  
Clears the H interrupt flip-flop for process A. This EOM  
should be issued every time an H interrupt is generated for process A.

\*\*\*\*\*  
EOM 30017B Clear H interrupt B

Clears the H interrupt flip-flop for process B. This EOM should be issued every time an H interrupt is generated for process B.

\*\*\*\*\*

EOM 30020B Clear H interrupt C

\*\*\*\*\*

Clears the H interrupt flip-flop for process C. This EOM should be issued every time an H interrupt is generated for process C.

\*\*\*\*\*

EOM 30021B Clear DWC $\emptyset$  C

\*\*\*\*\*

Clears the data word count  $\emptyset$  flip-flop for process C. This EOM should be issued everytime a data cycle operation occurs to permit subsequent data cycling. If a data cycle occurs while the process DWC $\emptyset$  flip-flop is set, the process is aborted (note the cycle operation does however occur).

\*\*\*\*\*

EOM 30022B Clear H interrupt D

\*\*\*\*\*

Clears the H interrupt flip-flop for process D. This EOM should be issued every time an H interrupt is generated for process D.

\*\*\*\*\*

EOM 30023B Clear DWC $\emptyset$  D

\*\*\*\*\*

Clears the data word count  $\emptyset$  flip-flop for process D. This EOM should be issued everytime a data cycle operation occurs

to permit subsequent data cycling. If a data cycle occurs while the process DWCØ flip-flop is set, the process is aborted (note the cycle operation does however occur).

\*\*\*\*\*

Enable the Hybrid Processor

\*\*\*\*\*

EOM\* 32244B

POT = 1

EOM\* 30244B

POT = 1

This will turn on the DSC-II and alert it to listen to Hybrid Processor requests.

\*\*\*\*\*

Test the Hybrid Processor

\*\*\*\*\*

The following sequence will test the DSC-II to make sure the "BBN MODS" switch is in the "mods ON" position.

EOM\* 32244B

POT = Ø

EOM\* 30244B

SKS\* 34000B

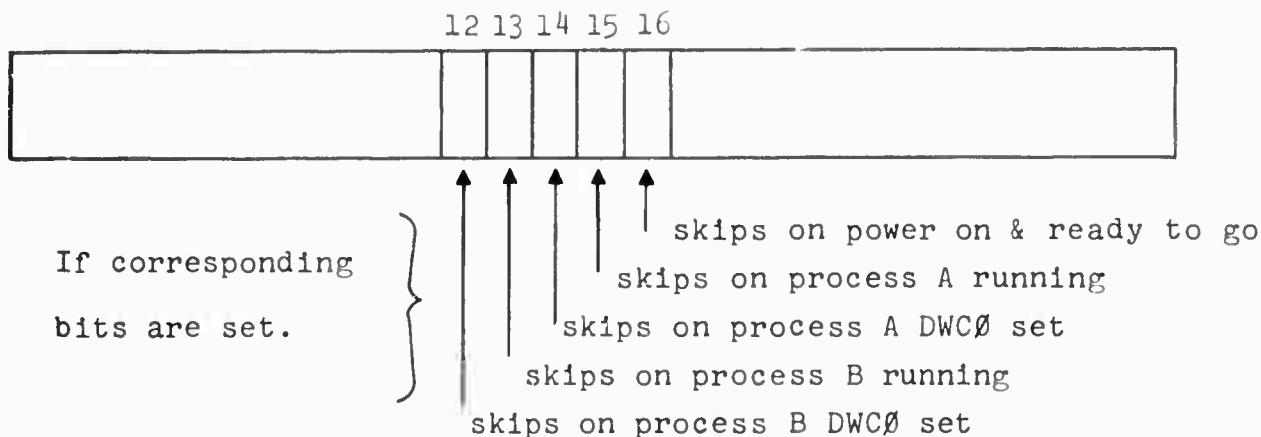
BRU \*+2

BRU TRAP (in wrong position)

An SKS to test the Hybrid Processor itself is available. Use of this SKS should be immediately preceded by an EOM\* 30244B. The SKS format is shown below.

SKS\* 3XYØ1B

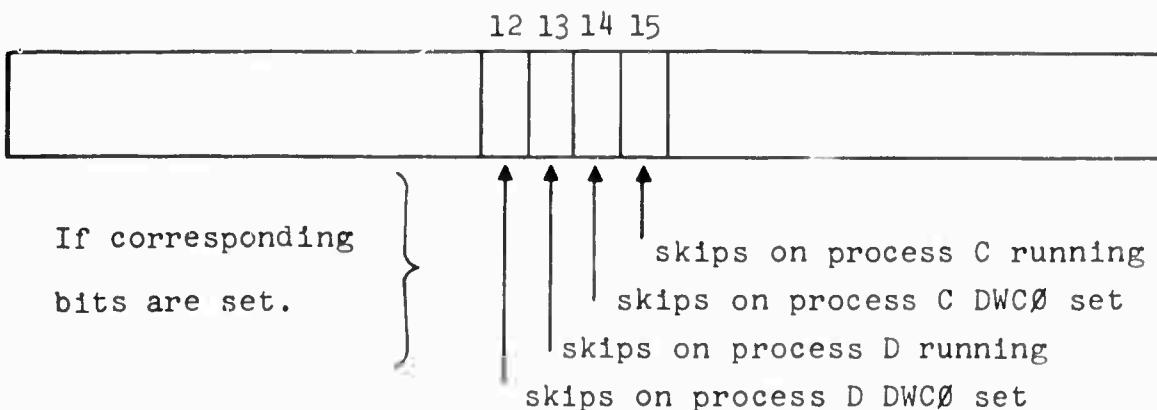
The XY field may test one or more of 5 indicators.



A second SKS tests 4 additional indicators.

SKS\* 3XY02B

The XY field is shown below:



If more than one indicator is tested, a skip will be generated if one or more of the tested indicators is set.

## A.2. Hybrid Processes

### A.2.1 Interlace Words

The location of the interlace words in core memory for each process is shown in Figure A-2.

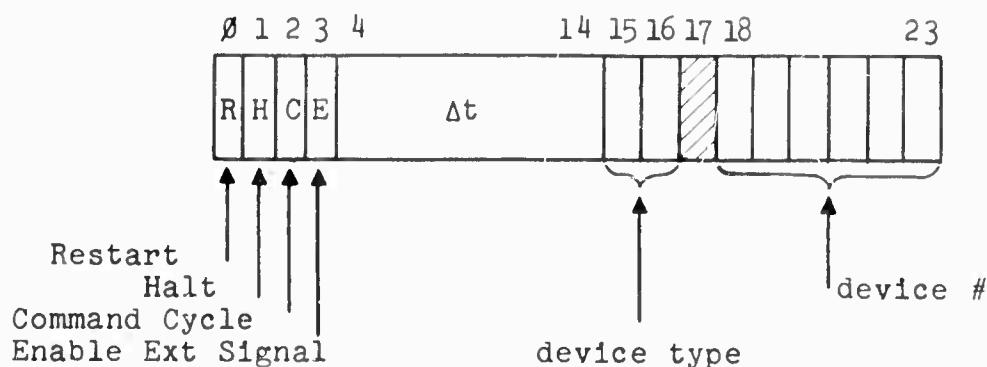
	Process A	Process B	Process C	Process D
current command interlace	240	244	250	254
current data interlace	241	245	251	255
cycle command interlace	242	246	252	256
cycle data interlace	243	247	253	257

FIGURE A-2 Location of Interlace Words

When a request for service is acknowledged, the current command interlace word is "updated" by adding 77600001B and storing the result; a command is now read from the absolute address specified by the rightmost 16 bits of the current command interlace word. The completion of the command triggers the updating (by adding 77600001B) of the current data interlace word and storing this result. The data is now read (or stored depending on the previous command) from the absolute address specified by the rightmost 16 bits of the current data interlace word.

When a command cycle operation occurs, the contents of the cycle command interlace word are read out then stored into the current command interlace word.

When a data cycle operation occurs the contents of the cycle data interlace word are read out. They are then stored into the current data interlace word. A process interrupt request is also generated during the data cycle operation. When this operation occurs for process A, the process A DWC0 flip-flop is set. Similarly for process B, the process B DWC0 flip-flop is set. If a new data-cycle operation is attempted and the process DWC0 flip-flop is set, the process is terminated after the cycle operation completes.

A.2.2 Command Format as Interpreted by the Hardware

The device type bits specify a category of devices with the device number bits specifying a device within a category. Table A-3 indicates the currently implemented devices and categories.

Bits 15 16

Ø	Ø	Analog to Digital Converter
Ø	1	Digital to Analog Converter
1	Ø	Digital to Digital Inputs
1	1	Digital to Digital Outputs

TABLE A-3 Device Types

The  $\Delta t$  field specifies the time between\* the execution of this command and the next command. This value is added to the process clock when the command is fetched from memory. If  $\Delta t = \emptyset$ , an immediate request for additional service for this process is generated, but a process sync pulse is not generated. If the sum of the process clock plus  $\Delta t$  is negative, an immediate request for additional service for this process is generated and a process sync pulse is generated. If the result of adding  $\Delta t$  to the process clock is positive, the process dismisses

\* The  $\Delta t$  field is ignored by unclocked processes.

itself until the process clock goes from the  $\emptyset$  to the -1 state at which time a request for service is generated and a process sync pulse is generated.

The "RHCE" bits are for control purposes and may be micro-programmed.

#### Halt Conditions

The Hybrid Processor will terminate service to a process under several conditions:

- a) The current command interlace word count (bits  $\emptyset$ -7) becomes  $\emptyset$ .
- b) The "H" (halt) bit is set in a command.
- c) The "E" bit is set in a command.
- d) The  $DWC\emptyset$  flip-flop was set when data cycling occurred.
- e) The clear process EOM was sent by the computer.

Conditions a), b), and c) can be over-ridden by setting the "R" (restart) bit in the same command word.

#### Interrupt Conditions

Each process has an associated priority interrupt channel.

<u>Clocked Processes</u>		<u>Unclocked Processes</u>	
Process A	Process B	Process C	Process D
2 $\emptyset$ 1	2 $\emptyset$ 2	2 $\emptyset$ 3	2 $\emptyset$ 4

An interrupt is generated for a process when any one or more of the following conditions is true:

- a) The H bit is set in a command word.
- b) The word count (bits  $\emptyset$ -7) of the current data interlace word becomes  $\emptyset$ .

Condition b) can be distinguished from a) because the current data interlace word count  $\emptyset$  condition sets the  $DWC\emptyset$  flip-flop for the process.

#### Synchronization to External Events

The "E" bit halts a process until a true signal is established on the process external signal input. The re-initiation of a process by this signal will clear the process clock\* and start it counting again which will initiate a request for service and generate a process sync pulse in 1 clock tick ( $10 \mu\text{sec}$ ). It is possible to use the "R" and "E" bits simultaneously to wait for a specified  $\Delta t$  or external signal, whichever occurs first.

#### A.3. Interrupts on the 940

A complete table of the interrupts assignments on the BBN-940 is listed here. The table is in order of decreasing priority.

31	TMCC W Buffer	WC $\emptyset$
33		EOR
56	Mem Parity	CPU
57	Mem Parity	IO
64	DACC	WC $\emptyset$
65	DACC	EOR
74 sync		Real Time Clock
75 pulse		
200	Alarm Clock	
201	Process A	
202	Process B	
203	Process C	
204	Process D	
205	Reserved	
206		
207	Anelex Line Printer	(WC $\emptyset$ )

(continued)

---

\* Applies only to clocked processes.

210	TTY Input
211	TTY Output
212	TTY Carrier On
213	TTY Carrier Off
214	
215	Reserved
216	
217	Hardwired On

Table A-4      BBN 940 Interrupts  
(numbers in octal)

#### A.4. Timing Considerations

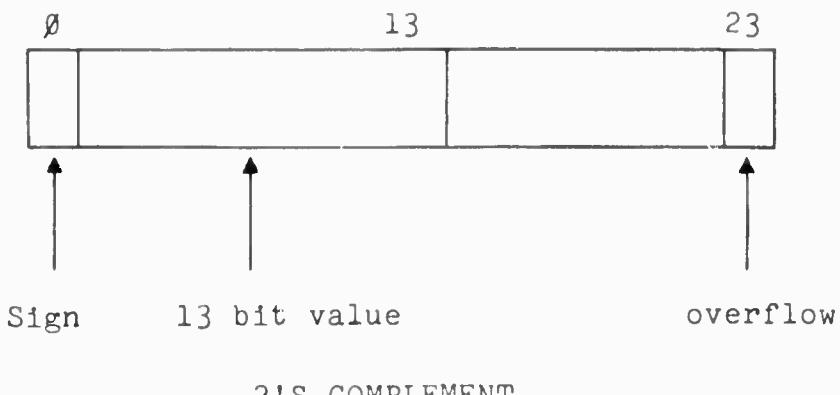
A Hybrid Processor normal operation (no command or data cycling or I/O waits) requires 6 memory accesses which can be executed in a minimum of 7 memory times. Measurements on the Hybrid Processor indicate that it can get these 6 memory accesses in about 11 memory times (i.e., 19  $\mu$ secs) because of DMC limitations. This means D/A operations can be performed at almost 50kHz. An I/O wait time is necessary with A/D conversions which limits A/D operations to 25kHz rates. Each cycling operation requires 2 additional memory accesses which normally take 4 memory times because of DMC limitations.

For consistent timing synchronization, interactions between the real-world and the 940 should be triggered by the process sync pulse. These pulses have a resolution of 10  $\mu$ sec. This synchronization is accomplished by causing the sample and hold operation of the appropriate A/D channels and the rank 1 to rank 2 D/A update to be triggered by the process sync pulse.

#### A.5. Details of the Built-In Systems

##### A/D Converter

The Adage A/D converter we are using has a 32 channel multiplexor. Eight of these channels have sample and hold amplifier inputs. The acquisition time of the sample and hold gate is 5  $\mu$ secs. The digital value returned by the A/D converter is in the following format.



Each channel accepts  $\pm 10$  volt signals and presents an input load of  $10K$  ohms.

#### D/A Converters

The 5 SDS D/A converter channels we are using accept 15 bit two's complement input values. Each channel has operational amplifier outputs with a .01 ohm output impedance and  $\pm 10$  volt signals. The conversion is stable after about 8  $\mu$ sec on large swings with smaller swings stabilizing sooner. Each channel has dual rank operation. The actual D/A conversion ladder is read out of the rank 2 flip-flops. The LOAD D/A pulses feed only the rank 1 flip-flops. Patching options permit the LOAD D/A pulses to trigger the rank 1 to rank 2 transfer or this transfer can be triggered by a process sync pulse.

It takes 4 commands to talk to the videsector. Two of these commands specify D/A conversions for establishing the videsector X,Y co-ordinates.

$-10$  volts       $X = 400000000B$       is the upper right hand corner of the videsector.

$-10$  volts       $Y = 400000000B$

$+10$  volts       $X = 37777777B$       is the lower left hand corner of the videsector.

$+10$  volts       $Y = 37777777B$

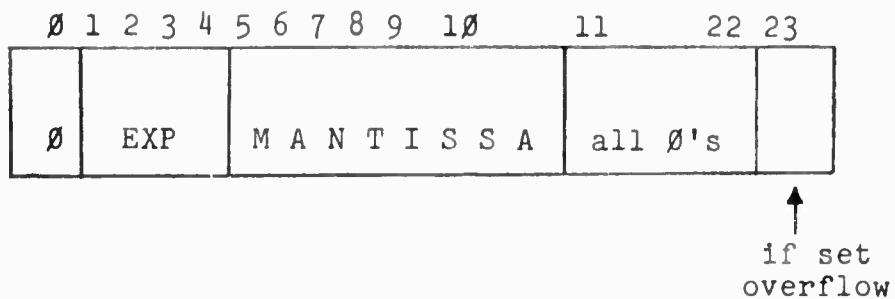
(These are the complement of the co-ordinates for the KM105 display scope.)

The command  $600B$  is a D to D output command which starts up the videsector. This command should be used with a wait for external signal specification. The command  $400B$  is a D to D input command which reads the videsector light value at the specified co-ordinates.

Typical Videsector Command Sequence

commands	data
$200B$	$X$
$6201B$	$Y$
$04000600B$	(ignored)
$00100400B$	light value

The value returned by the videsector is a left justified number (generally  $\log_2$ ).



These values are fully described in BBN Report #1537.

Note that you MUST PATCH the videsector completion pulse via a BNC cable to the external signal input of the process you are using to run the videsector.

Simple Scope Display

A simple scope (ITT KM105) is available on the 940 and is driven by D/A converters. The scope has 4 inputs X, Y, Z and INT. The 3 analog inputs are X, Y, Z.  $X, Y = -10$  is the lower left corner,  $X, Y = 10$  is the upper right corner.  $-10 < Z \leq 0$  shuts off the beam,  $0 < Z \leq 10$  turns the beam on with brightness increasing approximately linearly. A digital beam on enable input (INT) is

provided. This input is normally patched to the scope trigger output. The scope trigger input is normally patched to the load D/A pulse which specifies beam intensity. The trigger delays 20  $\mu$ sec then intensifies for 5  $\mu$ sec.

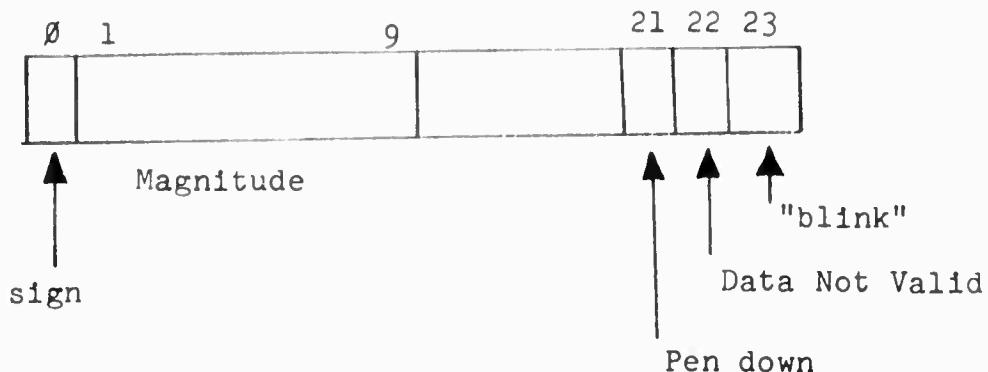
### Grafacon Interface

The BBN Grafacon 1010A (a version of the RAND Tablet) is interfaced to the Hybrid Processor in a manner to make it easy to use with the display. There are two pertinent commands:

Read Grafacon X Co-ordinate 402B

Read Grafacon Y Co-ordinate 403B

Each co-ordinate data word has the format below:



X = 37740000B

is the upper right hand corner of the tablet

Y = 37740000B

X = 40000000B

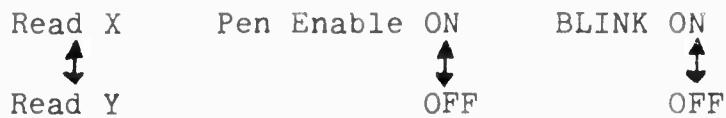
is the lower left hand corner of the tablet

Y = 40000000B

The pen down bit is a 1 if the pen switch is depressed, 0 if the pen switch is not depressed.

The data Not Valid bit is a 1 if the data is not valid, 0, if the data is valid. The Grafacon sets data not valid if two successive Grafacon cycles (~200  $\mu$ secs) indicate the pen has moved by more than 1 unit in either X or Y (such as if the PEN is up in the air too far away from the tablet).

The blink bit is a 1 if the data is not valid for either X or Y or if the pen is down for either X or Y. The use of the blink bit with displays will be explained below. Three switches affect the blink bit. These are labeled:



DO NOT CHANGE EITHER OF THE LEFT TWO SWITCHES while running a program using the Grafacon. (This can generate core memory parity errors.)

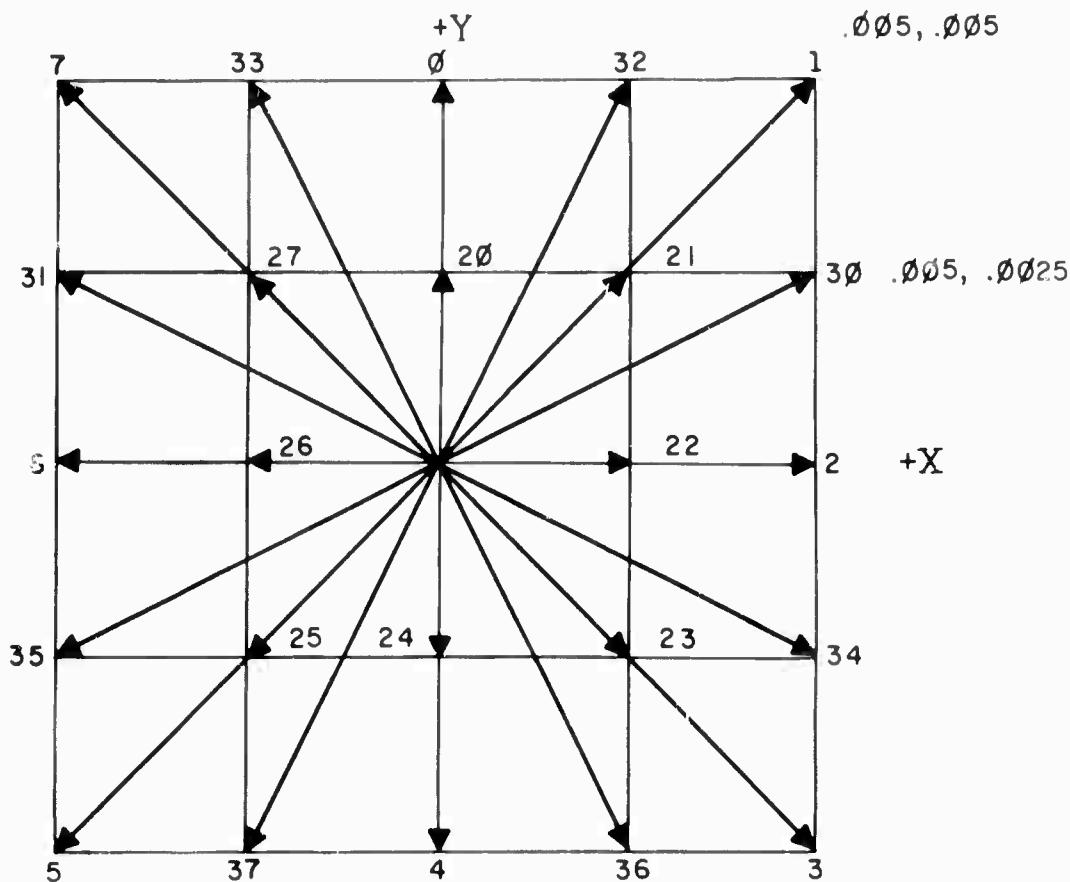
The purpose of all this complexity for bit 23 (blink) is to enable the non-display of particular Grafacon data points. For example, a data not valid point would not be displayed normally. Since Grafacon data tables are meant to be directly displayable, the use of the blink bit for points which shouldn't be displayed will indeed inhibit displaying if the blink switch is on. The read X/ read Y switch should be in the position of read X if the X data word is the display data word on which the display unblank is triggered or on read Y if the Y data word is the display data word on which display unblank is triggered. If you want the activation of the pen switch to blink the display, turn the pen enable on.

A sample of a Grafacon program to read data from the pen and display it directly is shown below.

@RESET.	115/	LDB =404	135/	LDB =600
@RECOVER FROM /GRAFACON/.	116/	BRS 205B	136/	BRS 204B
@CONTINUE DDT.	117/	Ø	137/	Ø
100/ LDX =-4	120/	Ø	140/	Ø
101/ LDA 404,2	121/	LDA =500	141/	CLAB
102/ BRS 201B	122/	LDB =600	142/	BRS 206B
103/ Ø	123/	BRS 204B	143/	EAX 1
104/ Ø	124/	Ø	144/	BRS 207B
105/ BRX 101	125/	Ø	145/	Ø
106/ CLX	126/	CLX	146/	Ø
107/ BRS 176B	127/	LDA =400	147/	Ø
110/ Ø	130/	LDB =402	150/	Ø
111/ EAX 1	131/	BRS 205B		
112/ BRS 176B	132/	Ø	400[	200
113/ Ø	133/	Ø	401[	100201
114/ LDA =402	134/	LDA =500	402[	402
			403[	3777403

Calcomp Plotter

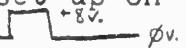
A model 665 Calcomp digital incremental plotter with a full step size of .005" is being interfaced to the Hybrid Processor. The plotter operates by moving a pen (ball-point or India ink) relative to the chart paper by stepping the pen carriage left or right (Y axis) and stepping the drum carrying the paper (X axis). One such step is initiated by doing HP command  $6048$ . The high 5 bits of the data specifies which of 24 possible steps is to be taken. The data value ( $X 2^{-19}$ ) and the corresponding plot movements are shown in the diagram below.



The pen may be lifted from the paper with data value  $11_8$   
 $X 2^{19}$  and put down with  $12_8 X 2^{19}$ .

The time between steps must be greater than 2.22 msec corresponding to a rate of 450 steps per second or 2.25 inches per second. In addition, after pen up or down commands, a delay must be observed to allow for the effects of the inertia of the pen. The interface monitors the commands given to the plotter and generates a holdoff signal whenever the plotter is not ready to accept the next command. The inverse of this signal is available for patching to the external signal inputs of an HP process. When used in this way, all timing is done by the interface and an unclocked HP process may be used.

#### Tektronix Storage Tube Display

Most oscilloscopes (including the storage tube display require a negative going signal to unblank.) (The KML05 uses a positive going signal.) A new pair of trigger, unblank BNC's has been set up on the patch panel to accept a standard SDS input pulse (  $\phi_v$ ) and output a negative going unblank pulse. The input should be patched to a load D/A pulse just as with the KML05. The storage tube unblank control is completely separate from the KML05 unblank control which permits running both scopes at once.

The storage scope requires an erase signal. This signal is generated by issuing command 603. The data word for this command is ignored. A BNC connector on the panel brings the erase level out for patching.

#### Applied Dynamics 4 Interface

Digital control over the Applied Dynamics 4 hybrid analog computer will soon be available via the Hybrid Processor. Three commands in the Hybrid Processor implement this control.

#### OUTPUT COMMANDS

- 602 - Clears out the AD-4 interface. The data word for this command is ignored. This command should be issued as the first command when you begin using the AD-4. It needn't be executed more than once per session.
- 601 - Output a command to the AD-4 interface. These command formats are described in the AD-4 interface spec. If the command specifies a subsequent input function, the wait for external signal bit should be used with the command (i.e. 04000601).

INPUT COMMANDS

The command 401 reads digital data from the AD-4 interface into the process's data table. The formats of this digital data are discussed in the AD-4 interface spec.

The completion pulse from the AD-4 interface should be patched to the external signal input of the process which is running the AD-4 interface.

Note that some of the output commands to the AD-4 interface specify a subsequent input function of an analog value. These commands cause the AD/4 to close the appropriate relays which place the selected analog value on a trunk which will be attenuated and patched to an A/D converter channel.

## APPENDIX B

## B. HYBRID PROCESSOR SOFTWARE

B.1. Structure of Software Interface

## B.1.1 Block Description

The fundamental unit of control in the Hybrid Processor software is a block. A block is a set of commands or data (not both) which are used sequentially one or more times by a single HP process. The size of a block of data is from 1 to 16K ( $K=1024_{10}$ ) words. The size of a block of commands is from 1 to a maximum determined by several factors which are described below. In the best case, a block of commands may currently be 2K words long.

Data blocks and command blocks are treated completely independently. The motivation for this is two-fold. First, a limited amount of space is available to store command blocks, and second, many uses of the HP (e.g., waveform generation, waveform sampling, and display generation) require relatively few unique commands which are used repeatedly over a large number of data words.

## B.1.1.1 Data Block Creation

A data block for use by the HP is created by means of a BRS\* which indicates to the monitor two addresses delimiting the area of the user's memory to be used for the block, the HP process number for which the block is being created, and the number of times the block is to be used (iteration count). The software assigns memory to contain the block to the user if necessary, places information about the block in a queue, and returns to the user's program a block number which is his handle on the block.

## B.1.1.2 Command Block Creation

A command block for use by the HP is similarly created by means of a BRS which indicates to the TSS monitor two addresses delimiting the area of the user's memory containing the commands

---

\* A BRS is a System Programmed Operator which traps to the monitor. The address of the BRS specifies the service of the monitor which is being invoked.

to be used in this block, the HP process number for which the block is being created, and the number of times the block is to be used (iteration count). The software first scans through the commands and checks that all the devices referenced by the commands are assigned to the user. Concurrently, the time required for the execution of a single iteration of the block is computed by summing the  $\Delta t$  information in the commands subject to modifications as described below. Since the time for a single iteration of a command block may be quite small, it is necessary that the command block be modified to avoid an excessive interrupt rate. This is accomplished by replicating the block of commands several times when the commands are copied into monitor memory. The commands thus formed will be referred to as an expanded command block. The number of these replications is determined as follows:

1. Compute the minimum number of iterations needed to exceed 2 msec. [MINIT]
2. If  $\text{MINIT} > (\text{number of iterations specified [SIT]})$  then the block is considered a fast block and is treated in a special manner.
3.  $\text{NIT} \leftarrow \text{MINIT}$
4. If  $(\text{SIT} \bmod \text{NIT}) > \text{MINIT}$ , or if  $(\text{SIT} \bmod \text{NIT}) = \emptyset$ , then we are done.
5. Else  $\text{NIT} \leftarrow \text{NIT} + 1$
6. Go to Step 4.

The total number of replications necessary is NIT. The execution of this block is started with the  $(\text{NIT} - (\text{SIT} \bmod \text{NIT}))$ -th replica and proceeds to the end thereby doing  $(\text{SIT} \bmod \text{NIT})$  iterations of the original command block. Thereafter, it is started at the first replica and NIT iterations are done per pass. If  $(\text{SIT} \bmod \text{NIT}) = \emptyset$ , the first pass starts with the first replica. The expanded block is executed N times such that  $(N-1) * \text{NIT} + (\text{SIT} \bmod \text{NIT}) = \text{SIT}$ . The minimum time per pass is  $(\text{SIT} \bmod \text{NIT})$  times the time for one iteration which is guaranteed by Step 4 above to be greater than 2 msec.

A fast block is a block for which the above procedure cannot possibly guarantee 2 msec per pass. In this case, the total time is made as long as possible, and a switch is set forbidding the subsequent command block from also being a fast block.

This may be done because there is sufficient buffering of interlace words to absorb a single fast block.

The number of replicas times the number of commands in the block yields the number of words needed in monitor memory to store the expanded block. Whenever any process is assigned to a user, a page for HP commands is acquired unless there is already such a page in existence. This page is locked into core memory and contains commands for all HP processes. The page exists as long as there is at least one process assigned to a user. Space in this 2K page is divided into 32 64-word chunks. An index table contains an entry for each of these 64 word chunks which is -1 if the space is free and a process number if it is assigned. A search is conducted through this index to find the smallest set of contiguous 64 word chunks which will hold the expanded command block. The commands are translated and placed at the beginning of this area, and the space used is assigned to the process.

In the event that sufficient space cannot be found, the BRS notifies the user program of this fact. Otherwise, a pointer to the initial entry into the expanded command block, a pointer to the beginning of the expanded command block, the modified iteration count (N), the block number, and the number of 64 word chunks used by this block are placed in the command block queue and the BRS returns the block number to the user's program.

#### B.1.2 Control Bits \*

The restart bit (R-bit, bit 0) is inserted into every command placed into monitor memory except if a command from user memory contains an H or E bit, (see below) the R bit is copied from the user's command. This effectively causes commands to be executed without regard to command word count.

The cycle bit (C-bit, bit 2) is completely masked out of the user's command, and a C bit inserted in the very last command in the expanded block. This causes a command cycle operation to occur only after the last command is done.

The user is allowed to place Halt bits (H-bit, bit 1) in his commands. The H bit will cause the process to stop until one of three possible conditions occur: the alarm clock reaches

---

\* See Appendix A for more information on control bits.

$\emptyset$  and is enabled for the process, or the command contains an E bit and the external signal is true, or the command contains an R bit. Because an H bit in a command causes an interrupt, the rate at which the process is restarted must be controlled. It is not possible to control the rate of an external signal, so it is declared illegal to have a command with both E and H bits. This is checked by the software. Similarly, a command is not allowed to contain both R and H bits. The software checks this too. The alarm clock rate is limited to control the remaining restart source.

Since the presence of an H bit is the only way to cause a command interrupt, an H bit must be inserted in the last command in the expanded block to produce an interrupt when a command cycle operation occurs. Thus it is necessary to count H interrupts to determine which one is due to a cycle operation. This count must be stored in the command block queue. To avoid lengthening the command block queue, the negative of this count is substituted for the first starting location of this block. In this case, the expanded block contains only one replica, and all starting locations are the same. This is possible because we have already controlled the interrupt rate and therefore only need one replica.

An External Signal Enable (E bit, bit 3) bit is copied directly into the expanded block, with the exception noted above. Its only other effect is to cause  $\Delta t$  to be considered  $\emptyset$  in determining the timing of a command block. This is due to the possibility of receiving an external signal immediately after the command is executed, thereby causing zero delay.

#### B.1.3 Data Look-Ahead

To avoid locking all memory pages containing data for all HP processes in core, a look-ahead scheme is employed in which the software looks ahead of the data currently being used and brings the data which will be needed into core memory (if necessary) and locks it. The amount of look-ahead is fixed at  $14000_8$  words. When the HP is finished with a data page, it is unlocked and may be written back on the drum as necessary. A further restriction on locking pages in memory is necessary since sufficient memory must be unlocked so the swapper can find room to load other programs which are running in the time-sharing system. This may reduce the look-ahead to less than  $14000_8$  words. If by the time the Hybrid Processor requires a data page, it is not locked in core, the process is aborted.

#### B.1.4 Reclamation of Space

Space used for commands, command block queues, and data block queues must be reclaimed for re-use after the HP is finished with the space. This is done by a phantom user to avoid excessive activity at the interrupt level.

### B.2. Programming for the Hybrid Processor

The previous portion of this appendix has discussed the operation of the monitor routines which interface the Hybrid Processor to the user's program. This section describes the calls to the monitor available to the user, and programming techniques using these monitor calls.

#### B.2.1 Calls to the Monitor

All explicit calls to the monitor in the SDS-940 time-sharing system are made by means of a SYStem Programmed OPerator or SYSPOP. There are 64 such SYSPOP's, each of which may communicate arguments to the monitor in any of the active registers, A, B, or X, or in the address of the SYSPOP. One SYSPOP called BRS (BRanch to System) uses its address field to multiplex many monitor calls into one SYSPOP. Several BRS's are used to communicate the user's intentions to the monitor HP interface. The BRS's used for this purpose are tabulated at the end of this appendix. The use of these BRS's is illustrated in the following sections.

#### B.2.2 Device Assignment

Usually all devices which a user plans to use should be assigned in the first part of his program. It is only necessary to assign devices once unless they are explicitly dismissed, or a RESET command is given to the EXEC. A typical method of assigning devices is to scan through the command block(s) and assign each device. This is illustrated in the following program excerpt:

•  
•  
•  
LUP1    LDX    =CTBL1-CTBL1E  
      LDA    CTBL1E,2  
      BRS    129  
      BRU    DNA                    DEVICE NOT AVAILABLE  
      BRU    ANA                    ALTERNATE DEVICE ASSIGNED  
      BRX    LUP1  
      LDX    =CTBL2-CTBL2E  
LUP2    LDA    CTBL2E,2  
      BRS    129  
      BRU    DNA  
      BRM    ANA  
      BRX    LUP2  
•  
•  
•  
CTBL1    DATA 200B,6201B,6202B,6201B,10203B  
CTBL2E    BSS Ø  
CTBL2    DATA 20B,21B,22B,23B,10024B  
CTBL2E    BSS Ø  
•  
•  
•

In cases where only a few devices are being assigned, each may be assigned separately as illustrated by the following code:

•  
•  
LDA    =200B  
BRS    129  
BRU    DNA  
BRU    ANA  
LDA    =201B  
BRS    129  
BRU    DNA  
BRU    ANA  
•  
•  
•

It should be noted that BRS 129 has 3 separate returns. It may skip 0, 1, or 2 times. If it skips twice, the assignment has been completely successful; once, an alternate device was assigned; zero, no device assigned. In the two cases where the requested device is not available, (0 or 1 skip) the job number of the user to whom the device is assigned is returned in the B register. This allows the user to discover who he must contact and where, if he wants the device.

A program excerpt to do this is given below:

```
        .
        .
        .
* IN B, THE JOB NUMBER OF THE USER TO WHOM THE
* DESIRED DEVICE IS ASSIGNED
* IN A, THE DEVICE DESIRED

STB      JOB
LDB      =8
LDX      =1
BRS      36      PRINT DEVICE NUMBER
           PRINT MESSAGE MACRO
MESSAGE (ASSIGNED TO)
LDA      JOB
BRS      144      TRANSLATE JOB NUMBER TO TTY NO.
STX      TTYNO    AND USER NUMBER
CAX
LDA      =(R)SS-1
CAB
BRS      98       NULL STRING POINTER
           TRANSLATE USER NUMBER TO NAME
HLT      IMPOSSIBLE
LDX      =1
BRS      35       PRINT NAME
MESSAGE (ON TELETYPE)
LDA      TTYNO
LDB      =10
LDX      =1
BRS      36      PRINT TTY NUMBER
        .
        .
        .
```

### B.2.3 Process Assignment

Two types of HP processes are available; clocked, and unclocked. A clocked process has an 11-bit clock which counts at a 100 kHz rate and is used to time the interval between the execution of successive commands. An unclocked process has no such clock and commands are executed as rapidly as possible unless timed with external signals, or the alarm clock. Command timing is discussed below.

Processes 0 and 1 are clocked processes. The only distinction between process 0 and process 1 is priority. When two processes make simultaneous requests for service, the lower numbered process wins. This means that process 0 is preferable for such application as waveform sampling where timing accuracy is important. For applications where timing accuracy is extremely important, the sample and hold gates should be used in conjunction with the process sync pulse as discussed in Appendix A.

Processes 2 and 3 are unclocked processes. Unclocked processes should be used for applications requiring no timing, such as one-shot sampling of several A/D converter channels, or those requiring low repetition rates which can be handled by the alarm clock, or those which are timed by external signals.

Processes are assigned using BRS 126 as follows:

LDX	PNO	PROCESS NUMBER IN X
BRS	126	
BRU	PNA	PROCESS NOT AVAILABLE
CXA		
SKE	PNO	
BRU	APA	ALTERNATE PROCESS ASSIGNED
.		
.		
.		

In the two cases where the assignment is not completely successful, the job number of the user to whom the desired process is assigned is returned in the B register. This allows the user to find out with whom he should bargain to get the process he desires. This procedure was discussed in section B.2.2.

#### B.2.4 Dismissing Device and Processes

It is important that when a user is finished with Hybrid Processor devices and processes, that he release their assignment so they are available to other users. In most cases, executing a RESET command in the EXEC is most convenient. In case the program must be preserved, and a RESET cannot be done, then the program should explicitly release devices and processes with BRS's 127, 128, 130, or 131.

#### B.2.5 Creating Command Blocks

In order to provide commands for the Hybrid Processor, one or more command blocks must be created using BRS 133. The commands in the block are checked for device assignment and timing restriction, and then translated and transferred to memory available only to the monitor. The location of the block in monitor memory, the block number, and the iteration count are placed in a queue which will be used on a first in, first out basis.

Since the commands are moved to monitor core when this BRS is done, it follows that the commands must be in the current memory space of the fork doing the BRS. However, since the block of commands is attached to a Hybrid Processor process, not to a fork, BRS 133 need not be executed in the same fork which runs the Hybrid Processor.

#### B.2.6 Creating Data Blocks

In order for the Hybrid Processor software to know what area(s) of a user's memory space to use for fetching and putting data, a data block for this purpose is established by means of BRS 132. The addresses delimiting the data block are placed in a queue along with the block number, and the iteration count. Blocks placed in this queue will be used on a first in, first out basis.

There are no restrictions on the length of a data block, however it should be pointed out that when the end of each block or a page boundary is reached, the CPU is interrupted. The time taken in the interrupt service routine is about 500  $\mu$ sec. Thus, if each of a series of blocks takes less than 500  $\mu$ sec to complete, the process will be aborted due to the inability to service interrupts rapidly enough. Furthermore, for times not much greater than 500  $\mu$ sec, a large portion of the CPU's time will be usurped for servicing interrupts, thus seriously

degrading service to all other activities in the time sharing system including any programs that are running for the user using the Hybrid Processor. The time used by the interrupt service routines is charged to the user who initiated its activity. For these reasons, multiple, short data blocks should be avoided.

A side effect of creating a data block is that the memory included in the block is assigned if necessary, to the fork doing the BRS. Except for this side effect, a data block may be created in any fork; however, the data will be taken from (put into) the memory space of the fork which starts the Hybrid Processor (BRS 134-135). The relationship of forks and the Hybrid Processor is discussed in more detail in a later section.

#### B.2.7 Constructing Command Blocks

A Command Block is nothing more than a sequence of Hybrid Processor commands which are to be executed by the Hybrid Processor. The length of a Command Block is restricted only by space and timing limitations. The best case maximum length is 2048 commands; the minimum time for all iterations of a Command Block is determined by the time it takes to respond to an interrupt from the Hybrid Processor. Failure to meet this minimum will result in multiple executions of the command block. There is no provision for detecting this.

#### B.2.8 Writing Hybrid Processor commands

A Hybrid Processor Command is a 24-bit SDS-940 machine word divided into 4 fields. These fields are listed below:

Control Field	bits 0-3 (bit 0 is leftmost bit of word)
Delta T Field	bits 4-14
Device Type	bits 15-16
Device Number	bits 18-23

Bit 17 is not used.

The device types are:

<u>Device Type</u>	<u>Command Code</u>	<u>Name</u>
00	0XX <sub>8</sub>	A/D
01	2XX <sub>8</sub>	D/A
10	4XX <sub>8</sub>	Digital Input
11	6XX <sub>8</sub>	Digital Output

where the x's indicate device number. There are 5 D/A converter channels (0-4), and 32 A/D multiplexor channels. Thus a typical D/A command might be 203<sub>8</sub>, and a typical A/D command might be 015<sub>8</sub>.

#### B.2.9 Timing of Hybrid Processor Commands

Hybrid Processor Commands may be timed in 3 ways: by using the delta t field of the commands, by using the external signal enable bit, or by using the alarm clock. The delta t field simply specifies the interval between the execution of one command and the next. Thus, if a command is to be executed 100  $\mu$ sec after the preceding command, then the delta t field of the preceding command should specify 100  $\mu$ sec. Since delta t is specified in terms of 10  $\mu$ sec ticks, 100  $\mu$ sec would correspond to a delta t of 10<sub>10</sub> or 12<sub>8</sub>. The sequence of two commands would appear as follows:

12203 <sub>8</sub>	D/A 3, 100 $\mu$ sec delta t
XX201 <sub>8</sub>	D/A 2, is executed 100 $\mu$ sec later

To use an external signal for timing, a command should contain the external signal enable bit (bit 3). This will cause the process to stop until the external signal for that process becomes true. In this case the commands would appear as follows:

4000203 <sub>8</sub>	D/A 3, wait for external signal
XXXX201 <sub>8</sub>	D/A 1, is executed upon receipt of ext signal

A variation on the above is possible by specifying both a wait for external and delta t.

44012203<sub>8</sub>      D/A 3 wait for ext signal or 100  $\mu$ sec

XXXXX201<sub>8</sub>      whichever is first.

In this case, the restart bit (bit 0) is set to one causing the clock to continue operating.

Two operations must be performed to use the alarm clock for timing. First, one or more commands must contain a halt bit (bit 1) and second, the rate at which the alarm clock generates signals must be specified when the process is started. When the use of the alarm clock is invoked, a signal is generated periodically which restarts the Hybrid Processor process unconditionally. This means that regardless of the state of the process, it will immediately do the next command. Thus, the process may be counting down a delta t, or waiting for an external signal, or halted due to the presence of a halt bit in the last command, and if the alarm clock produces a signal, the next command will be executed immediately. Normally, the process would be halted due to a halt bit, but the other cases are also possible. The specification of the alarm clock rate is discussed below. A command containing a halt bit appears as follows:

20000203<sub>8</sub>

#### B.2.10 Starting the Hybrid Processor

BRS 134 and BRS 135 are available for starting a Hybrid Processor process. They differ only in that BRS 134 returns immediately to the user's program whereas BRS 135 dismisses the user's program until the process is finished, or aborts prematurely. These BRS's take 3 arguments in X, A, and B. X contains the process number to be started. A contains a number which if  $> 10$ , and  $< 2^{23}/100_{10}$ , specifies the interval in 1 msec units between alarm clock signals, or if  $< 10$  that no alarm clock signals are to be generated. If A contains a number  $> 2^{23}/100$ , an illegal instruction trap will be generated. B contains a number which if  $> 0$  is the time of day at which the first command is to be executed, or if  $< 0$ , that the process is to start immediately. This time is in terms of clock ticks since the system was initialized modulo the number of ticks per day. To find the current time in these same units, do BRS 42. Thus to start a HP process, one second from now, the following code might be used:

```
        .
        .
        .
BRS    42    READ CURRENT TIME
ADD    =50    NUMBER OF TICKS PER SECOND
CAB
LDA    =<alarm clock interval> (OR CLA if no alarm clock)
LDX    PNO
BRS    134    OR BRS 135
        .
        .
        .
```

When the alarm clock is not being used, and it is desired to start the process immediately, the following code could be used.

```
        .
        .
        .
LDX    PNO
CLAB
BRS    134
        .
        .
        .
```

#### B.2.11 Synchronizing Program with Real-Time I/O

In most applications of real-time input/output, it is necessary to synchronize the action of the program with the I/O activity. This is necessary to insure that data to be output is prepared before it is output, and that input data is input before it is processed. The simplest way to synchronize program with I/O transfers is to prepare all output data before initiating the transfer, starting the transfer with a wait for completion (BRS 135), and then processing all the input data. Such a procedure is adequate only for those cases where the output data may be computed independently of the input data, and the total amount of data does not exceed the available memory space (16K). A second procedure is to stop the Hybrid Processor while processing data and then restarting it, repeating until the transfer is complete. Obviously, such a procedure will not allow real-time I/O for the whole transfer. A third method is provided by means of BRS 139 and BRS 140. These two BRS's dismiss the user's program until the completion of a specific data or command block.

BRS 139 takes as arguments the process number in X and the data block number in A. (Recall that BRS 132 returned the block number when the data block was created.)

#### B.2.12 Forks and the Hybrid Processor

Generally a Hybrid Processor process is attached to a time-sharing system job. A job consists of all the activity going on under one teletype for one user. When a user walks up to an inactive teletype and presses rubout, a new job is created, and when he EXIT's or LOG's OUT, the job disappears. Between these two acts, the job remains in existence. A job may include several forks, each of which may be running independently of all other forks subject to restrictions which will not be discussed here.

Because an HP process is attached to the job rather than a fork, several forks can be doing BRS's to govern the operation of the HP process. Thus, one fork may assign devices and processes, while another creates command blocks and so on. The instances in which an arbitrary fork can not be used are given below.

The memory space which is used for data during a Hybrid Processor transfer is established at the time the process is started (BRS 134 or BRS 135) and consists of the memory in the relabeling of the fork doing the BRS 134 or BRS 135. This means that if a data block is created from another fork with a different relabeling, the data will not be taken from (put into) the memory space of the fork creating the data block. Also, if the fork that starts the HP process changes its relabeling, the change will not be communicated to the running HP process. The relabeling of a fork may change explicitly via BRS's 4,44,121 or implicitly through the automatic acquisition of memory by referencing an unassigned page. An example of programming which will not work is given on the next page.

<u>Instructions</u>	<u>Fork Relabeling</u>	<u>HP Relabeling</u>
.	61000000,00000000	non-existent
.		
LDA =1024		
LDB =4096		
BRS 132		
.	61620000,00000000	non-existent
.		
BRS 134	61620000,00000000	61620000,00000000
.		
.		
LDA =4096		
LDB =12288		
BRS 132		
.	61626364,65660000	61620000,00000000
.		
.		

Note that when the fork does BRS 134, only the first 2 pages of memory are assigned. Memory assigned for the second data block does not exist as far as the HP process is concerned and when HP process attempts to use the second data block, it will abort.

When a command block is created, the commands are immediately taken from the current memory space of the fork doing the BRS 133. Thus, the commands must exist in the memory space of the fork doing the BRS. However, all command blocks do not have to be created in one fork; some may be created in one fork and others in other forks.

If the fork which started an HP process aborts for any reason (memory panic, rubout, illegal instruction, etc.) the process is also aborted as if a BRS 138 had been done.

#### B.3. Table of Hybrid Processor's BRS's

(Following on next page.)

11Dec67

G-1-2

BRS 126D

Assign Process

accepts in X

a virtual process number

BRS 126D

returns caller+1

if no process of the type requested  
is available

returns in B

job number of job to which the process  
is assigned

returns caller+2

process available

returns in X

actual process number assigned

returns in B

if alternate process assigned, job number

Restrictions: The process number must be a valid process  
number.

This BRS considers processes in two classes; clocked and  
unclocked. Process numbers 0 and 1 are clocked processes,  
and 2 - 3 are unclocked. If the process requested is not  
available, another process of the same class will be  
assigned if available.

BRS 127D

Dismiss Process

accepts in X

a virtual process number

BRS 127D

The process is no longer assigned. If the process was  
running, it is stopped.

Restrictions: X must contain a valid process number.

11Dec67

G-2-1

BRS 128D Dismiss All Processes

BRS 128D

Does BRS 127D for all processes.

BRS 129D Assign Device

accepts in A 15-16 Device type

00	A/D
01	D/A
10	digital input
11	digital output

accepts in A 18-23 Virtual Device number

BRS 129D

returns caller+1 Nothing available

in B Job number of job with the device assigned.

returns caller+2 Alternate device assigned

in B Job number of job with the device assigned

returns in A Physical device actually assigned

returns caller+3 Requested device assigned

This BRS considers devices in classes as follows:

000 - 037<sub>8</sub> straight A/D

200<sub>8</sub>-204<sub>8</sub> D/A

All others are unique. If the requested device is not available, a search is made through the rest of the class to which it belongs and if one is free, it is assigned instead. Reference to a device is always made by virtual device number and this is translated to real device number.

27Sept67

0-3-0

\*\*\*\*\*

BRS 130D                      Dismiss Device

\*\*\*\*\*

accepts in A                      virtual device as for BRS 129D

BRS 130D

The device is de-assigned. Any processes running for this user are stopped, and all command and data blocks deleted. If the device was not assigned, this instruction is treated as a NOP.

\*\*\*\*\*

BRS 131D                      Dismiss all Devices

\*\*\*\*\*

BRS 131D

This does BRS 130D for all devices.

27 Sept 67

G-4-8

BRS 132D	Create Data Block
accepts in A	address of first word of data block
in B	address of first word after data block
in X	virtual process number
BRS 132D	
accepts caller+1	iteration count $N > 0$ iterate N times
	iteration count $N \leq 0$ iterate until stopped.
returns caller+2	No room for data block
returns caller+3	OK
returns in A	data block number

This BRS sets up a block of data to be used by an HP process. More than one data block can be created in which case when a block is completed data from the next is used and so on. Data blocks may be created at any time and will be used in the order created. When all data blocks have been used, the process will be terminated. Dismissing, assigning, or stopping a process will cause all data blocks to be deleted.

Restrictions: A, B must specify a non-negative non-zero length block. X must contain an assigned process number. Memory included in the block will be assigned by this BRS, however the memory available to an HP process is determined when the process is started. If a block is created which references memory which did not exist when the process was started, that memory will not be available to the HP and the process will stop when it gets to the point of using it. Care should be exercised to prevent successive appearances of short data blocks or data blocks with page boundaries near either end.

27Sept67

0-5-0

XX

BRS 133D Create Command Blocks

XX

accepts in A address of first command

in B address of first word after all commands

in X process number

BRS 133D

accepts in caller+1 iteration count as for BRS 132D

returns caller+2 No room

returns caller+3 OK

returns in A command block number

This BRS is similar to BRS 132D except it operates with blocks of commands instead of data. This BRS translates and transfers the commands into protected monitor memory. Each command is checked to ascertain if devices are assigned.

Restrictions: Same as BRS 132, plus all commands must use only assigned devices. Furthermore, care should be exercised to specify a block whose execution time (for all iterations) is not excessively short (1 millisecond is a good minimum).

27Sept67

G-6-3

\*\*\*\*\*

**BRS 134D Start Process and Continue**

\*\*\*\*\*

accepts in X      process number

    in A      alarm clock interval (10  $\mu$ sec per  
                  count) >100 if <100 alarm clock  
                  not used

    in B      time of day when to start (milliseconds)

    BRS 134

**Restrictions:** The process must be assigned and not running, there must be at least one data block and at least one command block. The HP must be on, and the DSCII must be in new mode, otherwise this BRS is illegal. If the relabelling of the process contains any shared or read only memory, the shared memory will not exist for the HP.

\*\*\*\*\*

**BRS 135D Start Process and Dismiss**

\*\*\*\*\*

accepts in A/B/X      same as for BRS 134D

    BRS 135D

    returns caller+1      when the process terminates.

**Restrictions:**      same as for BRS 134D

\*\*\*\*\*

**BRS 136D Dismiss Until Process Completes**

\*\*\*\*\*

accepts in X      process number

    BRS 136

    returns caller+1      when process is no longer running

**Restrictions:**      Process must be assigned.

BRS 137D Skip if Process Is Not Running

accepts in X process number

BRS 137D

returns caller+1 if process is running

                  caller+2 if process is not running or not assigned

Restrictions: Process must be valid.

BRS 138D Stop Process

accepts in X process number

BRS 138D

The process is stopped, all data and command blocks are deleted, memory locked for use by the HP is unlocked.

\*\*\*Note that if the fork which started a process panics or is terminated for any reason (rubout,etc.) the process is stopped, but data and command blocks are not deleted, and memory is not unlocked. If the process is started (BRS 134, BRS 135) it will continue where it left off except that time is reinitialized.

Restrictions: X must contain a valid process number.

27Sept67

6-8-8

\*\*\*\*\*

**BRS 139D** Dismiss Until Data  
Block is Finished

accepts in X process number

in A data block number (from BRS 132D)

BRS 139D

Program is de-activated until all iterations of the  
specified data block have been completed.

\*\*\*\*\*

**BRS 140D** Dismiss Until Command  
Block is Finished

accepts in X process number

in A command block number (from BRS 133D)

BRS 140D

Program is de-activated until all iterations of the  
specified command block have been completed.

\*\*\*\*\*

**BRS 141D** Read Current Blocks

\*\*\*\*\*

accepts in X process number

BRS 141D

returns in A current command block number

in B current data block number

in X current data address.

(real core address, only low 11  
bits are meaningful)

27 Sept 67

0-9-0

\*\*\*\*\*

BRS 142D Terminate Current Data Block

\*\*\*\*\*

accepts in X process number

BRS 142D

The iteration counter for the current data block is set to 1. This specifies the current iteration to be the last.

\*\*\*\*\*

BRS 143D Terminate Current Command Block

\*\*\*\*\*

accepts in X process number

BRS 143D

The iteration counter for the current command block is set to 1.

\* TYPE OUT THE MESSAGE "FILE NAME IS"

START LDA =MESS  
BRM TOS

\* OPEN THE FILE SPECIFIED FROM TELETYPE TO BE READ

CLEAR  
BRS 15  
BRU START  
BRS 16  
0  
STA FILE

\* READ THE DATA FROM THE SPECIFIED FILE INTO CORE

LDX =DTABLE-EDTBLE  
START1 WIO FILE  
CLB  
LSH 10  
STA EDTBLE,2  
BRX START1

\* CLOSE THE FILE

LDA FILE  
BRS 20

\* TYPE OUT THE MESSAGE "ACTUAL PROCESS ASSIGNED = "

LDA =MESS1  
BRM TOS

\* ASK FOR PROCESS CLOCK 0 (HIGHEST PRIORITY)

CLX  
BRS 126

- \* IF NO PROCESS CLOCK AVAILABLE HALT

0

- \* TYPE OUT THE ACTUAL PROCESS # ASSIGNED

CXA  
LDB =10  
EAX 1  
BRS 36

- \* TYPE OUT THE MESSAGE "ACTUAL D TO A CHANNEL = "

LDA =MESS2  
BRM TOS

- \* ASK FOR D TO A CHANNEL 0

LDA CTBLE  
BRS 129

- \* IF NO D TO A CHANNEL AVAILABLE HALT

0

- \* IF ALTERNATE AVAILABLE ACCEPT IT

NOP

- \* TYPE OUT THE ACTUAL D TO A CHANNEL ASSIGNED

ETR =77B  
LDB =10  
EAX 1  
BRS 36  
BRM CRLF

- TELL THE MONITOR THE BEGINNING AND END OF THE DATA TABLE

```
LDA =DTABLE
LDB =EDTABLE
CLX
BRS 132
```

- INDEFINITE NUMBER OF ITERATIONS

```
0
```

- HALT IF NO ROOM LEFT IN MONITOR FOR THESE TABLE POINTERS

```
0
```

- TELL THE MONITOR THE BEGINNING AND END OF THE COMMAND TABLE

```
LDA =CTBLE
LDB =CTBLE+1
BRS 133
```

- SPECIFY INDEFINITE NUMBER OF ITERATIONS

```
0
```

- HALT IF NO ROOM LEFT IN MONITOR FOR COMMAND TABLE

```
0
```

- START HYBRID PROCESSOR IMMEDIATELY

```
CLAB
BRS 135
BRU START
```

- MESSAGE TYPING ROUTINES

TOS 0  
LDB =-1  
EAX 1  
BRS 34  
BRR TOS

CRLF 0  
TCO =155B  
TCO =152B  
BRR CRLF

\* TEMPORARY STORAGE

FILE 0

\* TEXT STRINGS

MESS ASC '\$FILE NAME IS /'  
MESS1 ASC '\$ACTUAL PROCESS ASSIGNED = /'  
MESS2 ASC '\$ACTUAL D TO A CHANNEL = /'

\* BOUNDS OF DATA TABLE

DTABLE EQU 500B  
EDTBLE EQU 37777B

\* COMMAND TABLE FOR USER D TO A CHANNEL 0, DELTA T OF 40 MICROSECONDS

CTBLE DATA 4200B

END

APPENDIX C

C. HARDWARE PATCH PANEL

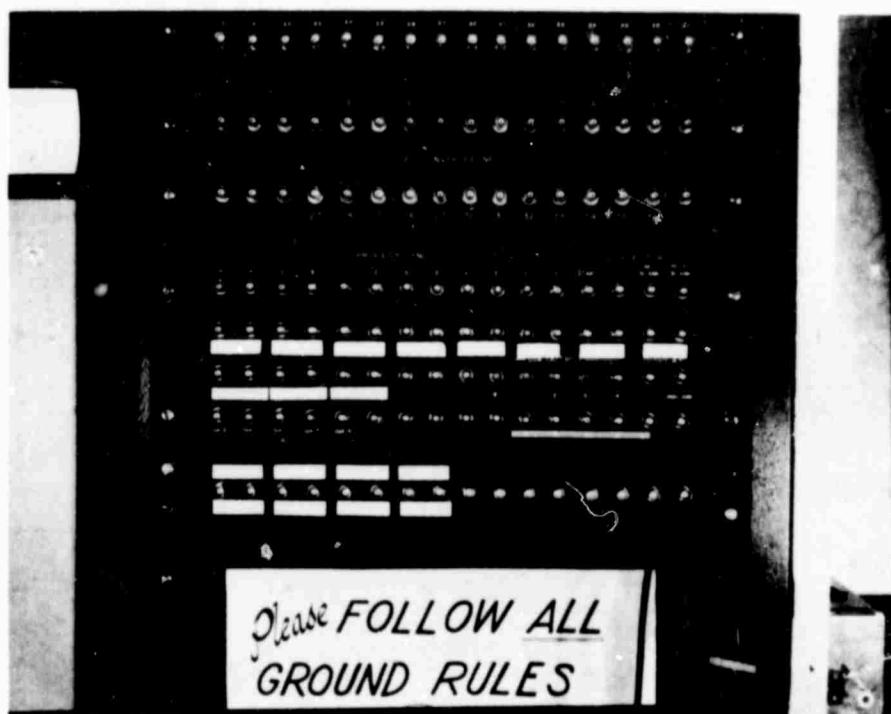


Figure C-1

The Hybrid Processor patch panel serves two major functions:

- 1) Patchable attachments to real world devices to and from the Hybrid Processor's inputs and outputs.
- 2) Patchable timing synchronization pulses.

### C.1. Patching to the Hybrid Inputs/Outputs

Since the Hybrid Processor is a time-shared device, hybrid devices must attach themselves via those access ports which are currently available. For example, the simple display scope requires three analog inputs. The D/A output of the D/A channel which is assigned to the user for the X co-ordinate should be patched to the X scope input cable, the D/A output of the D/A channel which is assigned for the Y co-ordinate should be patched to the Y scope input cable, the D/A output of the D/A channel assigned for the Z co-ordinate should be patched to the Z scope input cable. Patching is accomplished by attaching the cables via BNC connectors to the labeled patch panel connectors.

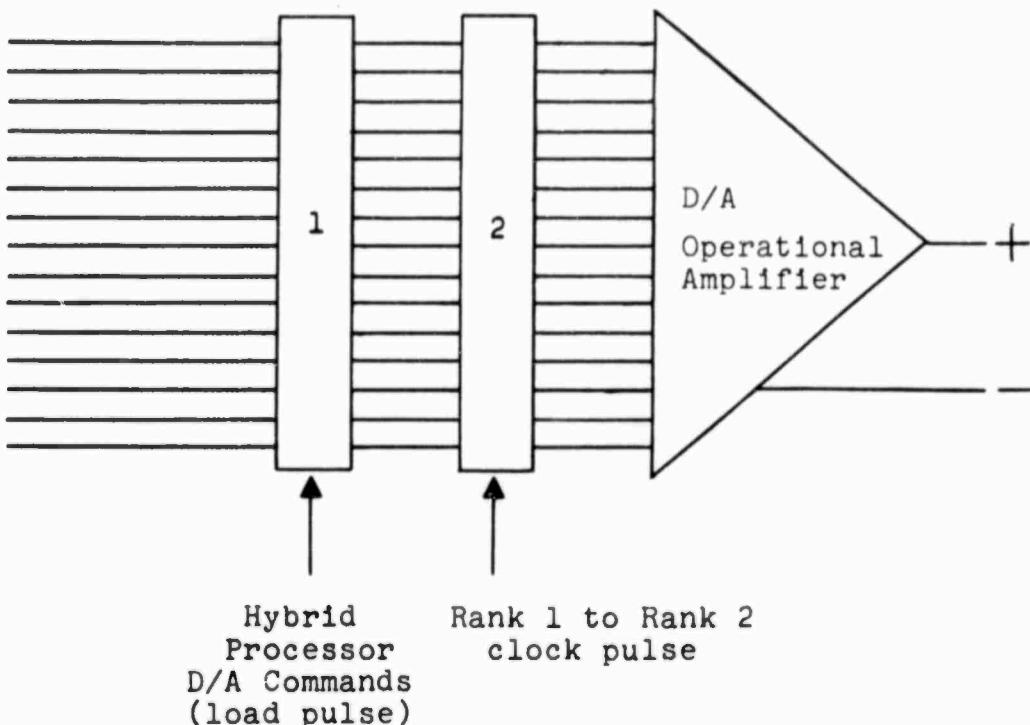
The inputs to the A/D converter must be patched in the same way. The analog signal cables must be patched to the labeled A/D channel to which they have been assigned. Note that the A/D channels are labeled in octal from  $\emptyset$  - 37.

### C.2. The Timing Sync Pulses

When a user is assigned to a clocked process, he has available to him a very accurately timed pulse when the clock goes from the  $\emptyset$  to the -1 state. Process  $\emptyset$  corresponds to sync pulse A while process 1 corresponds to sync pulse B. Both polarities of the sync pulses are available. The unbarred (e.g., A) sync pulse makes a ground to +8 volts transition (but not +8 to ground) at precisely the time when the clock counts to -1 while the barred (e.g., A) sync pulse makes a +8 volts to ground transition at precisely the same time.

### C.3. Patching D/A Converters

Our D/A converters are known as dual rank converters. This means each D/A converter has two associated flip-flop registers.



The D/A output is read out of the rank 2 flip-flop register. The D/A commands from the Hybrid Processor normally affect only the rank 1 flip-flop registers and thus do not change the actual D/A output. The purpose of this dual rank operation is to permit the sequential loading of several rank 1 registers, then later providing a single accurately timed pulse to move all the rank 1 values to the rank 2 registers thereby updating all the D/A outputs simultaneously. This rank 1 to rank 2 pulse can be provided by the clock pulse. Note that the move pulse must go from plus 8 volts to ground to cause the move. Therefore, it is convenient to use the barred clock pulse (such as A or B) in the D/A clock inputs.

When D/A update timing is non-critical (as when the D/A converters are driving the display), it is convenient to view the D/A converters as a single rank device. This may be accomplished by patching the individual D/A load pulse to its clock input. Now the Hybrid Processor D/A commands will set both ranks simultaneously and will therefore cause immediate D/A conversions.

#### C.4. Patching A/D Converters

The inverse of the dual-rank D/A operation is available for A/D conversions. That is, it is possible to sample and hold the analog voltages to be A/D converted before the actual conversion is performed. This necessary since conversions are performed sequentially and each conversion takes finite time. The sample and hold operation can be triggered indirectly by the process clock sync pulse. Since the aperture time of our sample and hold gates is 5  $\mu$ secs, the unbarred process clock sync pulse must be extended by a patchable one-shot to 5  $\mu$ secs. The output of this one-shot is then patched to one or more sample and hold--hold gate inputs. The analog signal to be sampled is patched to the appropriate analog S and H gate input. The corresponding S and H gate output is then patched to the assigned A/D converter input.

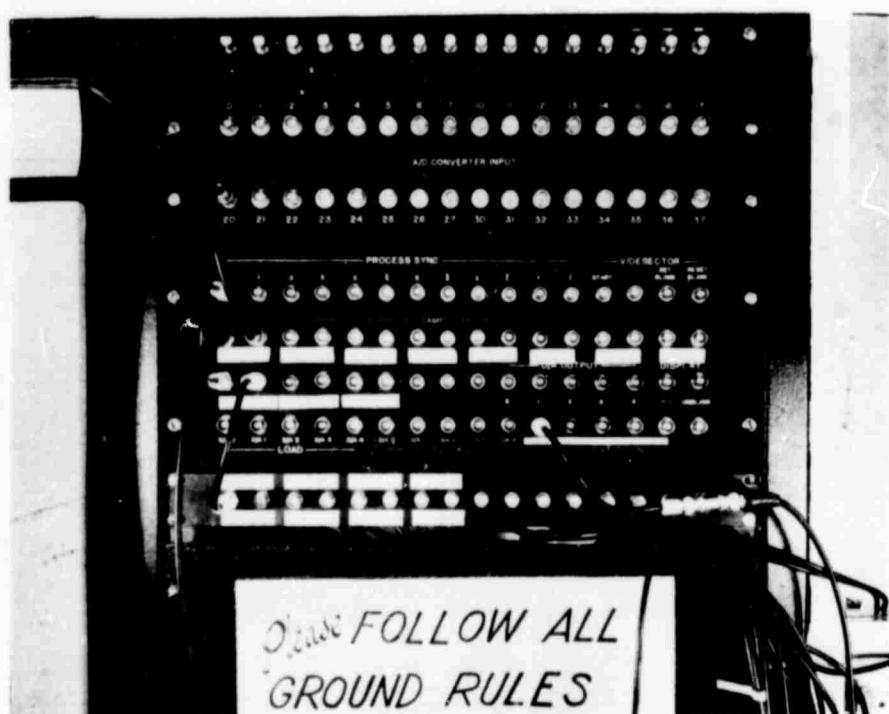


Figure C-2

PATCHING TO A SAMPLE AND HOLD INPUT

### C.5 The Display Scope

Most of the information for patching the display scope has been provided in C.1 and C.3. The patching of the INT trigger input remains to be discussed. A conventional method of running the simple display involves presenting an X, Y, and Z command each time a point is displayed. This suggests that the display trigger input be attached to load Z D/A pulse. (See Figure C-3) The display trigger output is attached to the display unblank input.

It is equally acceptable to give only X and Y commands to the display. (In fact, if intensity variation is not essential, the display can be run from only two D/A converters leaving the Z input floating.) In this case, either the load X or the load Y (whichever is given last and has a sufficient 40 usec time delay) pulse may be patched to the Z trigger input.

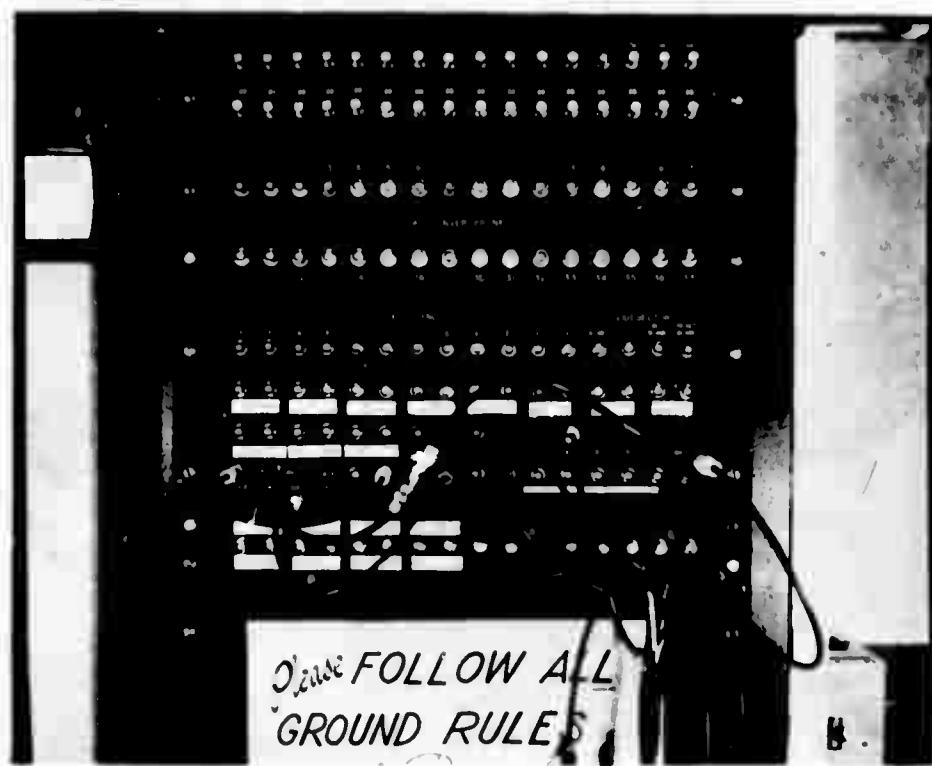


Figure C-3  
CONVENTIONAL DISPLAY PATCHING

### C.6. Patching to External Signal Inputs

It is possible to cause a Hybrid Process to pause until an external signal is presented. Each process has its own labeled external signal input. When the signal on this input is held at ground (at least 7 milliamps), the external signal is considered false. When this signal presents a high input impedance or rises to +8 volts (no more than 12 volts or damage could result), the external signal is considered true. The duration of the external signal at the true level must be  $>2 \mu\text{sec}$  to be recognized.

A convenient source of the external signal input is the gate output of the model 453 Tektronix scope with a parallel 680 resistor to -25 volts. This is convenient because the ordinary Tektronix scope trigger input accepts a wide variety of signal input shapes and amplitudes.

### C.7. Grounds

When attaching equipment to the Hybrid Processor, the user must be very careful about his grounding procedure. Any improperly grounded equipment can introduce noise in the entire analog portion of the Hybrid Processor. Generally, this noise is not insignificant.

The following ground rules must be followed by all users attaching equipment to the Hybrid Processor.

1. Equipment must not be connected to any grounds other than as stated below. This means no power line 3rd wire, no racks leaning against other racks, etc.
2. Signal ground at one point in user's apparatus should be connected with heavy copper braid to the Hybrid Processor analog ground. This is a large chunk of copper in the rear of the patch panel rack. The braid should be encased in insulating tubing to prevent spurious grounding.
3. A/D inputs: the shield of the cable should be connected to analog ground at the signal source.
4. D/A outputs: the shield is already connected to analog ground at the Hybrid Processor. It should not be connected to anything at the user's apparatus. In case

this is inconvenient, the shield may be grounded at both ends in which case the cable(s) should be run parallel to the ground braid in a bundle to minimize the induction of voltage in the loop thus created.

5. Digital signals: if the apparatus uses any digital signals from or to the Hybrid Processor, the return for these signals should be via a separate path to digital ground. Under no circumstances should the analog ground be connected to digital ground. Digital ground appears in the following locations on the patch panel.
  1. shield on all external signal inputs.
  2. shield on all single shot inputs and outputs.
  3. shield on D/A load and clock signals.
  4. shield on process sync pulse outputs.
  5. shield on display "trigger" and "unblank."
  6. shield on S and H hold inputs.
6. The user's apparatus should not inject excessive current into the analog ground. Some equipment is wired with a capacitor between the chassis and power line. This equipment should be avoided.

## REFERENCES

1. R. Belluardo, R.E. Gocht, and G.A. Paquette, "The Hybrid Computation Facility at United Aircraft Corporation Research Laboratories," Proceedings, DECUS (Digital Equipment Computer Users Society), 1963, Maynard, Mass., pp. 261-269, 1964.
2. Connelly, Mark, "Preliminary Design of a Hybrid Interface for a Time-Shared, Real-Time Simulation Facility," Memorandum No. 2, M.I.T. ESL-DSR 76259, 30 January 1968.
3. Connelly, Mark, "Preliminary Design of a Time-Shared, Real-Time Simulation Facility," Memorandum No. 1. M.I.T., ESL-DSR 76259, 19 December 1966.
4. Fiala, E.R., M.I.T. Masters Thesis, "Scheduling of Real-Time Processes in a Time-Shared Environment," 1968.

## Unclassified

Security Classification

## DOCUMENT CONTROL DATA - R &amp; D

(Security classification of title body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate or bat)		2a. REPORT SECURITY CLASSIFICATION
Bolt Beranek and Newman Incorporated 50 Moulton Street Cambridge, Massachusetts 02138		Unclassified
2b. GROUP		

3. REPORT TITLE

THE BBN HYBRID PROCESSOR

4. DESCRIPTIVE NOTES (Type of report and, inclusive dates)

Scientific. Interim.

5. AUTHORISI (First name, middle initial, last name)

Theodore R. Strollo Edward R. Fiala  
Raymond S. Tomlinson Jerome I. Elkind

6. REPORT DATE 30 June 1968	7a. TOTAL NO OF PAGES 93	7b. NO OF REFS 4
8a. CONTRACT OR GRANT NO ARPA Order No. 627 F19628-68-C-0125	9a. ORIGINATOR'S REPORT NUMBER(S) BBN Report No. 1686 Scientific Report No. 7	
8b. PROJECT NO 8668	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) AFCRL-68-0381	
c. DoD Element 6154501R		
d. DoD Subelement n/a		

10. DISTRIBUTION STATEMENT

Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce, for sale to the general public.

11. SUPPLEMENTARY NOTES This research was sponsored by the Advanced Research Projects Agency.	Air Force Cambridge Research Laboratories (CRB) L. G. Hanscom Field Bedford, Massachusetts 01730
--	--

12. ABSTRACT

Real-time experiments present special problems to a time-shared computer system. For example, simulations, waveform analysis, waveform generations, and displays require very high-rate, closely timed input-output between the digital computer's core memory and external devices. In many cases, conventional input/output methods fail to satisfy the requirements of the experiment. For this reason, Bolt Beranek and Newman Inc., has developed the concept of a special purpose "Hybrid Processor" to perform real-time input/output.

This report describes the hardware and software for the Hybrid Processor which has been constructed for BBN's SDS-940 computer, and the revisions planned for the Hybrid Processor which will be designed for the PDP-10.

**Unclassified**

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Hybrid Processor						
Real Time						
Time Sharing						
Hybrid Interfaces						